Facultad de Ciencias, UNAM
Lenguajes de Programación
Tarea 5

Rubí Rojas Tania Michelle

07 de diciembre de 2020

1. Evalúa la siguiente expresión usando el tipo de alcance y régimen de evaluación que se indica. Es necesario incluir el ambiente final en forma de pila en cada caso.

```
{with {a {+ 2 2}}
   {with {b {+ a a}}
      {with {foo {fun {x} {- x b}}
         {with {a {- 2 2}}
            {with {b {- a a}}
               {foo -3}}}}}}}
```

   a) Alcance estático y evaluación glotona
   Solución: La expresión que debemos evaluar es `{foo -3}`, por lo que

$$\begin{aligned}
\texttt{\{foo -3\}} &= \texttt{\{\{fun \{x\} \{- x b\}\} -3\}} \\
&= \texttt{\{- (-3) b\}} \\
&= \texttt{\{- (-3) 8\}} \\
&= -11
\end{aligned}$$

b	0
a	0
foo	[closureV: x,{-x b}, env-ant:((x -3),(b 8),(a 4))]
b	8
a	4

Tabla 1: Ambiente final

   b) Alcance dinámico y evaluación glotona
   Solución: La expresión que debemos evaluar es `{foo -3}`, por lo que

$$\begin{aligned}
\texttt{\{foo -3\}} &= \texttt{\{\{fun \{x\} \{- x b\}\} -3\}} \\
&= \texttt{\{- (-3) b\}} \\
&= \texttt{\{- (-3) 0\}} \\
&= \texttt{-3}
\end{aligned}$$

x	-3
b	0
a	0
foo	{fun {x} {- x b}}
b	8
a	4

Tabla 2: Ambiente final

c) Alcance estático y evaluación perezosa
SOLUCIÓN: La expresión que debemos evaluar es `{foo -3}`, por lo que

```
{foo -3} = {{fun {x} {- x b}} -3}
         = {- (-3) b}
         = {- (-3) {+ a a}}
         = {- (-3) {+ {+ 2 2} {+ 2 2}}}
         = {- (-3) {+ 4 4}}
         = {- (-3) 8}
         = -11
```

b	{- a a}
a	{- 2 2}
foo	[closureV: x,{-x b}, env-ant:((x -3),(b {+ a a}), (a {+ 2 2}))]
b	{+ a a}
a	{+ 2 2}

Tabla 3: Ambiente final

d) Alcance dinámico y evaluación perezosa
SOLUCIÓN: La expresión que debemos evaluar es `{foo -3}`, por lo que

```
{foo -3} = {{fun {x} {- x b}} -3}
         = {- (-3) b}
         = {- (-3) {- a a}}
         = {- (-3) {- {- 2 2} {- 2 2}}}
         = {- (-3) {- 0 0}}
         = {- (-3) 0}
         = -3
```

x	-3
b	{- a a}
a	{- 2 2}
foo	{fun {x} {- x b}}
b	{+ a a}
a	{+ 2 2}

Tabla 4: Ambiente final

2. Dada la siguiente función:

```
(define (goo l)
    (if (empty? l)
        empty
        (append (car l) (goo (cdr l)))))
```

a) Explica qué hace y dale un nombre mnemotécnico.
SOLUCIÓN: Por cómo está definida la función `goo`, ésta debe recibir una lista de listas; por lo que `goo` hará la concatenación de las listas de la lista l, es decir, regresa una lista con todos los elementos de las listas de la lista l de acuerdo a su órden de aparición en su respectiva lista. Así, un nombre mnemotécnico para esta función podría ser `concatena-listas-de-lista`. Por lo tanto, nuestra función queda de la siguiente forma:

```
(define (concatena-listas-de-lista l)
    (if (empty? l)
        empty
        (append (car l) (concatena-listas-de-lista (cdr l)))))
```

b) Muestra los registros generados cuando es llamada con el argumento `'((1 2) (3 4) (4 6))`. ¿Cuántos registros son generados? ¿Cuántos son ocupados a la vez?

SOLUCIÓN: Veamos que se generan 8 registros de activación (4 de entrada y 4 de salida), y ningún registro es ocupado a la vez.

Ingresa `(concatena-listas-de-lista '((1 2) (3 4) (4 6)))`

```
(append '(1 2) (concatena-listas-de-lista '((3 4) (4 6))))
                    (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                 l = '((1 2) (3 4) (4 6))
                concatena-listas-de-lista
```

Ingresa `(concatena-listas-de-lista '((3 4) (4 6)))`

```
   (append '(3 4) (concatena-listas-de-lista '((4 6))))
                    (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                  l = '((3 4) (4 6))
                concatena-listas-de-lista
```

```
(append '(1 2) (concatena-listas-de-lista '((3 4) (4 6))))
                    (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                 l = '((1 2) (3 4) (4 6))
                concatena-listas-de-lista
```

Ingresa `(concatena-listas-de-lista '((4 6)))`

```
      (append '(4 6) (concatena-listas-de-lista '()))
                    (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                      l = '((4 6))
                concatena-listas-de-lista
```

```
   (append '(3 4) (concatena-listas-de-lista '((4 6))))
                    (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                  l = '((3 4) (4 6))
                concatena-listas-de-lista
```

```
(append '(1 2) (concatena-listas-de-lista '((3 4) (4 6))))
                    (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                 l = '((1 2) (3 4) (4 6))
                concatena-listas-de-lista
```

Ingresa `(concatena-listas-de-lista '())`

```
                          '()
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                        l = '()
               concatena-listas-de-lista
```

```
     (append '(4 6) (concatena-listas-de-lista '()))
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                     l = '((4 6))
               concatena-listas-de-lista
```

```
   (append '(3 4) (concatena-listas-de-lista '((4 6))))
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                   l = '((3 4) (4 6))
               concatena-listas-de-lista
```

```
(append '(1 2) (concatena-listas-de-lista '((3 4) (4 6))))
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                 l = '((1 2) (3 4) (4 6))
               concatena-listas-de-lista
```

Sale `(concatena-listas-de-lista '())`

```
                  (append '(4 6) '())
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                     l = '((4 6))
               concatena-listas-de-lista
```

```
   (append '(3 4) (concatena-listas-de-lista '((4 6))))
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                   l = '((3 4) (4 6))
               concatena-listas-de-lista
```

```
(append '(1 2) (concatena-listas-de-lista '((3 4) (4 6))))
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                 l = '((1 2) (3 4) (4 6))
               concatena-listas-de-lista
```

Sale `(concatena-listas-de-lista '((4 6)))`

```
                (append '(3 4) '(4 6))
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                   l = '((3 4) (4 6))
               concatena-listas-de-lista
```

```
(append '(1 2) (concatena-listas-de-lista '((3 4) (4 6))))
                 (if (empty? l) empty
  (append (car l) (concatena-listas-de-lista (cdr l))))
                 l = '((1 2) (3 4) (4 6))
               concatena-listas-de-lista
```

Sale `(concatena-listas-de-lista '((3 4) (4 6)))`

```
(append '(1 2) '(3 4 4 6))
(if (empty? l) empty
(append (car l) (concatena-listas-de-lista (cdr l))))
l = '((1 2) (3 4) (4 6))
concatena-listas-de-lista
```

Sale `(concatena-listas-de-lista '((1 2) (3 4) (4 6)))`

```
'(1 2 3 4 4 6)
```

c) Optimiza la función usando la técnica de recursión de cola.

SOLUCIÓN:

```
(define (concatena-listas-de-lista l)
  (concatena-listas-de-lista-tail l '()))

(define (concatena-listas-de-lista-tail l acc)
  (if (empty? l)
      acc
      (concatena-listas-de-lista-tail (cdr l) (append acc (car l)))))
```

d) Muestra los registros generados por la función del inciso anterior con el argumento `'((1 2) (3 4) (4 6))`. ¿Cuántos registros son generados? ¿Cuántos son ocupados a la vez?

SOLUCIÓN: Veamos que se generan 6 registros de activación y que 4 de estos registros son ocupados a la vez.

Entra `concatena-listas-de-lista '((1 2) (3 4) (4 6)))`

```
(concatena-listas-de-lista-tail '((1 2) (3 4) (4 6)) '())
(concatena-listas-de-lista-tail l '())
l = '((1 2) (3 4) (4 6))
concatena-listas-de-lista
```

Ingresa/Sale `(concatena-listas-de-lista-tail '((1 2) (3 4) (4 6)) '())`.

```
(concatena-listas-de-lista-tail '((3 4) (4 6)) '(1 2))
(if (empty? l) acc
(concatena-listas-de-lista-tail (cdr l) (append acc (car l))))
l = '((1 2) (3 4) (4 6)), acc = '()
concatena-listas-de-lista-tail
```

Ingresa/Sale `(concatena-listas-de-lista-tail '((3 4) (4 6)) '(1 2))`

```
(concatena-listas-de-lista-tail '((4 6)) '(1 2 3 4))
(if (empty? l) acc
(concatena-listas-de-lista-tail (cdr l) (append acc (car l))))
l = '((3 4) (4 6)), acc = '(1 2)
concatena-listas-de-lista-tail
```

Ingresa/Sale `(concatena-listas-de-lista-tail '((4 6)) '(1 2 3 4))`

```
(concatena-listas-de-lista-tail '() '(1 2 3 4 4 6))
(if (empty? l) acc
(concatena-listas-de-lista-tail (cdr l) (append acc (car l))))
l = '((4 6)), acc = '(1 2 3 4)
concatena-listas-de-lista-tail
```

Ingresa/Sale `(concatena-listas-de-lista-tail '() '(1 2 3 4 4 6))`

```
                    '(1 2 3 4 4 6)
                  (if (empty? l) acc
(concatena-listas-de-lista-tail (cdr l) (append acc (car l))))
              l = '(), acc = '(1 2 3 4 4 6)
             concatena-listas-de-lista-tail
```

Obtenemos

```
'(1 2 3 4 4 6)
```

3. Dada la siguiente función

```
(define (foo n)
    (if (< n 10)
        n
        (+ (modulo n 10) (foo (quotient n 10)))))
```

a) Explica qué hace y dale un nombre mnemotécnico.

SOLUCIÓN: La expresión `(modulo n 10)` regresa el residuo de la división de n entre 10; en particular, como la división es entre 10, entonces regresará el último digito del número n. La expresión `(quotient n 10)` regresa el resultado de dividir a n entre 10; en particular, como la división es entre 10, elimina el último dígito del número n, Así, `foo` regresa la suma de los dígitos de n; por lo que un nombre mnemotécnico para esta función podría ser `suma-digitos`. Por lo tanto, la función queda de la forma:

```
(define (suma-digitos n)
  (if (< n 10)
      n
      (+ (modulo n 10) (suma-digitos (quotient n 10)))))
```

b) Muestra los registros generados cuando es llamada con el argumento 1729. ¿Cuántos registros son generados? ¿Cuántos son ocupados a la vez?

SOLUCIÓN: Veamos que se generan 8 registros de activación y que ningún registro es ocupado a la vez.

Ingresa `(suma-digitos 1729)`

```
             (+ 9 (suma-digitos 172))
                  (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 1729
                   suma-digitos
```

Ingresa `(suma-digitos 172)`

```
             (+ 2 (suma-digitos 17))
                  (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 172
                   suma-digitos
```

```
             (+ 9 (suma-digitos 172))
                  (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 1729
                   suma-digitos
```

Ingresa `suma-digitos 17)`

```
(+ 7 (suma-digitos 1))
    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
        n = 17
    suma-digitos
```

```
(+ 2 (suma-digitos 17))
    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
        n = 172
    suma-digitos
```

```
(+ 9 (suma-digitos 172))
    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
        n = 1729
    suma-digitos
```

Ingresa `(suma-digitos 1)`

```
1
    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
        n = 1
    suma-digitos
```

```
(+ 7 (suma-digitos 1))
    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
        n = 17
    suma-digitos
```

```
(+ 2 (suma-digitos 17))
    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
        n = 172
    suma-digitos
```

```
(+ 9 (suma-digitos 172))
    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
        n = 1729
    suma-digitos
```

Sale `(suma-digitos 1)`

```
                    (+ 7 1)
                 (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 17
                  suma-digitos
```

```
               (+ 2 (suma-digitos 17))
                    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 172
                  suma-digitos
```

```
               (+ 9 (suma-digitos 172))
                    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 1729
                  suma-digitos
```

Sale `(suma-digitos 17)`

```
                    (+ 2 8)
                 (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 172
                  suma-digitos
```

```
               (+ 9 (suma-digitos 172))
                    (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 1729
                  suma-digitos
```

Sale `(suma-digitos 172)`

```
                    (+ 9 10)
                 (if (< n 10) n
(+ (modulo n 10) (suma-digitos (quotient n 10))))
                     n = 1729
                  suma-digitos
```

Sale `(suma-digitos 1729)`

c) Optimiza la función usando la técnica de recursión de cola.
SOLUCIÓN:

```
(define (suma-digitos n)
  (suma-digitos-tail n 0))

(define (suma-digitos-tail n acc)
  (if (= n 0)
      acc
      (suma-digitos-tail (quotient n 10) (+ acc (modulo n 10)))))
```

d) Muestra los registros generados por la función del inciso anterior con el argumento 1729. ¿Cuántos registros son generados? ¿Cuántos son ocupados a la vez?

SOLUCIÓN: Veamos que se generan 7 registros de activación y que 5 de estos registros son ocupados a la vez.

Ingresa `(suma-digitos 1729)`

```
(suma-digitos-tail 1729 0)
 (suma-digitos-tail n 0)
        n = 1729
      suma-digitos
```

Ingresa/Sale `(suma-digitos-tail 1729 0)`

```
                (suma-digitos-tail 172 9)
                     (if (= n 0) acc
(suma-digitos-tail (quotient n 10) (+ acc (modulo n 10))))
                    n = 1729, acc = 0
                   suma-digitos-tail
```

Ingresa/Sale `(suma-digitos-tail 172 9)`

```
                (suma-digitos-tail 17 11)
                     (if (= n 0) acc
(suma-digitos-tail (quotient n 10) (+ acc (modulo n 10))))
                    n = 172, acc = 9
                   suma-digitos-tail
```

Ingresa/Sale `(suma-digitos-tail 17 11)`

```
                (suma-digitos-tail 1 18)
                     (if (= n 0) acc
(suma-digitos-tail (quotient n 10) (+ acc (modulo n 10))))
                    n = 17, acc = 11
                   suma-digitos-tail
```

Ingresa/Sale `(suma-digitos-tail 1 18)`

```
                (suma-digitos-tail 0 19)
                     (if (= n 0) acc
(suma-digitos-tail (quotient n 10) (+ acc (modulo n 10))))
                    n = 1, acc = 18
                   suma-digitos-tail
```

Ingresa/Sale `(suma-digitos-tail 0 19)`

```
                           19
                     (if (= n 0) acc
(suma-digitos-tail (quotient n 10) (+ acc (modulo n 10))))
                    n = 0, acc = 19
                   suma-digitos-tail
```

Sale `(suma-digitos 1729)`

```
19
```

4. Dada la siguiente función

```
(define (hoo n l)
    (if (zero? n)
        l
        (hoo (sub1 n) (cdr l))))
```

a) Explica qué hace y dale un nombre mnemotécnico.

SOLUCIÓN: La función `hoo` elimina los primeros n elementos de la lista l, por lo que un nombre mnemotécnico podría ser `elimina-n-elementos`. Así, la función quedaría como

```
(define (elimina-n-elementos n l)
    (if (zero? n)
        l
        (elimina-n-elementos (sub1 n) (cdr l))))
```

b) Muestra los registros generados cuando es llamada con los argumentos 3 y `'(1 2 3 4)`. ¿Cuántos registros son generados? ¿Cuántos son ocupados a la vez?

SOLUCIÓN: Veamos que se generan 8 registros de activación, y ninguno de estos registros son ocupados a la vez.

Ingresa `(elimina-n-elementos 3 '(1 2 3 4))`

```
(elimina-n-elementos 2 '(2 3 4))
        (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
        n = 3, l = '(1 2 3 4)
         elimina-n-elementos
```

Ingresa `(elimina-n-elementos 2 '(2 3 4))`

```
(elimina-n-elementos 1 '(3 4))
        (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
        n = 2, l = '(2 3 4)
         elimina-n-elementos
```

```
(elimina-n-elementos 2 '(2 3 4))
        (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
        n = 3, l = '(1 2 3 4)
         elimina-n-elementos
```

Ingresa `(elimina-n-elementos 1 '(3 4))`

```
(elimina-n-elementos 0 '(4))
      (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
       n = 1, l = '(3 4)
      elimina-n-elementos
```

```
(elimina-n-elementos 1 '(3 4))
      (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
       n = 2, l = '(2 3 4)
      elimina-n-elementos
```

```
(elimina-n-elementos 2 '(2 3 4))
      (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
       n = 3, l = '(1 2 3 4)
      elimina-n-elementos
```

Ingresa `(elimina-n-elementos 0 '(4))`

```
'(4)
      (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
       n = 0, l = '(4)
      elimina-n-elementos
```

```
(elimina-n-elementos 0 '(4))
      (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
       n = 1, l = '(3 4)
      elimina-n-elementos
```

```
(elimina-n-elementos 1 '(3 4))
      (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
       n = 2, l = '(2 3 4)
      elimina-n-elementos
```

```
(elimina-n-elementos 2 '(2 3 4))
      (if (zero? n) l
(elimina-n-elementos (sub1 n) (cdr l)))
       n = 3, l = '(1 2 3 4)
      elimina-n-elementos
```

Sale `(elimina-n-elementos 0 '(4))`

```
'(4)
```

c) Optimiza la función usando la técnica de recursión de cola.
SOLUCIÓN:

d) Muestra los registros generados por la función del inciso anterior con los argumentos 3 y `'(1 2 3 4)`. ¿Cuántos registros son generados? ¿Cuántos son ocupados a la vez?
SOLUCIÓN:

5. Evalúa la siguiente expresión usando el intérprete para cajas visto en clase. Debes usar alcance estático y evaluación glotona. Mostrar el ambiente (stack) y memoria (heap) finales.

```
{with {a {newbox 17}}
   {with {b {newbox 29}}
      {with {foo {fun {} {setbox a {openbox a}}}}
         {with foo {fun {} {setbox b {openbox a}}}}
            {seqn {foo}
               {+ {openbox a} {openbox b}}}}}}
```

SOLUCIÓN: La expresión que debemos evaluar es

```
{seqn {foo}
  {+ {openbox a} {openbox b}}}
```

Como se trata de una expresión `seqn`, entonces primero debemos evaluar la función `foo`. De esta forma,

$$\texttt{\{foo\}} = \texttt{\{fun \{\} \{setbox b \{openbox a\}\}}}$$

Como `foo` no recibe ningún parámetro, entonces no agregamos nada al stack, sólo evaluamos `{setbox b {openbox a}}`. Esta expresión nos indica que a la caja b le debemos asignar el valor que contiene la caja a. Luego, evaluamos y regresamos el valor de `{+ {openbox a} {openbox b}}`, ya que es nuestra última expresión dentro de `seqn`. Así,

$$\begin{aligned}\texttt{\{+ \{openbox a\} \{openbox b\}\}} &= \texttt{\{+ 17 17\}} \\ &= 34\end{aligned}$$

Por lo tanto, el resultado de evaluar nuestra expresión es 34. Además, el stack y el heap quedan de la siguiente forma:

foo	0x13
foo	0x12
b	0x11
a	0x10

Tabla 5: Ambiente (stack)

0x13	[closureV: ,{setbox b {openbox a}} env-ant: env3]
0x12	[closureV: ,{setbox a {openbox a}} env-ant: '((17) (17))]
0x11	17
0x10	17

Tabla 6: Memoria (heap)

6. Dada la definición de la función `next-location` que genera nuevas direcciones de memoria, vista en clase, modifícala para que no tenga efectos secundarios, es decir, que no dependa de ninguna variable externa.

   Hint: Modifícala usando la técnica Store Passing Style

```
(define last-location -1) ;; al inicio del intérprete

;; next-location: number
(define (next-location)
   (begin
      (set! last-location (add1 last-location))
      last-location))
```

SOLUCIÓN: Modificamos el intérprete para lograr esto.

```
(define (next-location expr)
  (type-case Store expr
    [mtSto () 0]
    [aSto (location value expr)
          (+ 1 location)]))
```