

Fundamentos de lenguajes de programación

Semántica de los Conceptos Fundamentales de Lenguajes de Programación

Facultad de Ingeniería. Universidad del Valle

Marzo de 2018

Contenido

- 1 Un interpretador más complejo
- 2 Evaluación de expresiones condicionales
- 3 Ligadura local

Contenido

- 1 Un interpretador más complejo
- 2 Evaluación de expresiones condicionales
- 3 Ligadura local

Contenido

- 1 Un interpretador más complejo
- 2 Evaluación de expresiones condicionales
- 3 Ligadura local

Contenido

- 1 Un interpretador más complejo
- 2 Evaluación de expresiones condicionales
- 3 Ligadura local

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:

Valor Expresado = Número + Booleano

Valor Denotado = Número + Booleano

Un interpretador más complejo

Gramática

La gramática para el lenguaje será la siguiente:

$\langle \text{programa} \rangle ::= \langle \text{expresión} \rangle$

a-program (exp)

$\langle \text{expresión} \rangle ::= \langle \text{número} \rangle$

lit-exp (datum)

$::= \langle \text{identificador} \rangle$


var-exp (id)


$::= \langle \text{primitiva} \rangle (\{ \langle \text{expresión} \rangle \}^*(,))$

primapp-exp (prim rands)

Un interpretador más complejo

Gramática

$::=$  `if <expresión> then <expresión> else <expresión>`
`if-exp (test-exp true-exp false-exp)`

$::=$  `let {<identificador> = <expresión>}* in <expresión>`
`let-exp (ids rands body)`

$\langle\text{primitiva}\rangle ::= + \mid - \mid * \mid \text{add1} \mid \text{sub1}$

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)))
```

Especificación de la Gramática

```
(define grammar-simple-interpreter
  ((program (expression) a-program)
   (expression (number) lit-exp)
   (expression (identifier) var-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)  
  (subtract-prim)  
  (mult-prim)  
  (incr-prim)  
  (decr-prim))
```

Contenido

- 1 Un interpretador más complejo
- 2 Evaluación de expresiones condicionales
- 3 Ligadura local

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 no tener que definir un nuevo tipo de dato que maneje booleanos, falso se representará con cero y cualquier otro valor representará verdadero (como en C).

Semántica de los condicionales

Para esto se implementa la función `true-value?`. Esta función recibe un argumento y determina si corresponde al valor booleano falso (es igual a cero) o al valor booleano verdadero (cualquier otro valor).

```
(define true-value?  
  (lambda (x)  
    (not (zero? x))))
```

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 (true-value? (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 $*(y, 10)$

(Note: Red arrows point from the text above to the subexpressions: one to $-(x, 4)$, one to $+(y, 11)$, and one to $(y, 10)$.)*

- Primero se evalúa la subexpresión $-(x, 4)$. Dado que x vale 4 en el ambiente en el que se evalúa la expresión, el valor de la expresión es 0.
- Como el valor de la subexpresión $-(x, 4)$ es 0, se evalúa la subexpresión $*(y, 10)$.
- Finalmente, el valor de toda la expresión if es 20.

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 evalúa la subexpresión $-(x,4)$. Dado que x vale 4 en el ambiente en el que se evalúa la expresión, el valor de la expresión es 0.
- Como el valor de la subexpresión $-(x,4)$ es 0, se evalúa la subexpresión $*(y,10)$.
- Finalmente, el valor de toda la expresión `if` es 20.

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 evalúa la subexpresión $-(x,4)$. Dado que x vale 4 en el ambiente en el que se evalúa la expresión, el valor de la expresión es 0.
- Como el valor de la subexpresión $-(x,4)$ es 0, se evalúa la subexpresión $*(y,10)$.
- Finalmente, el valor de toda la expresión `if` es 20.

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 $*(y,10)$`

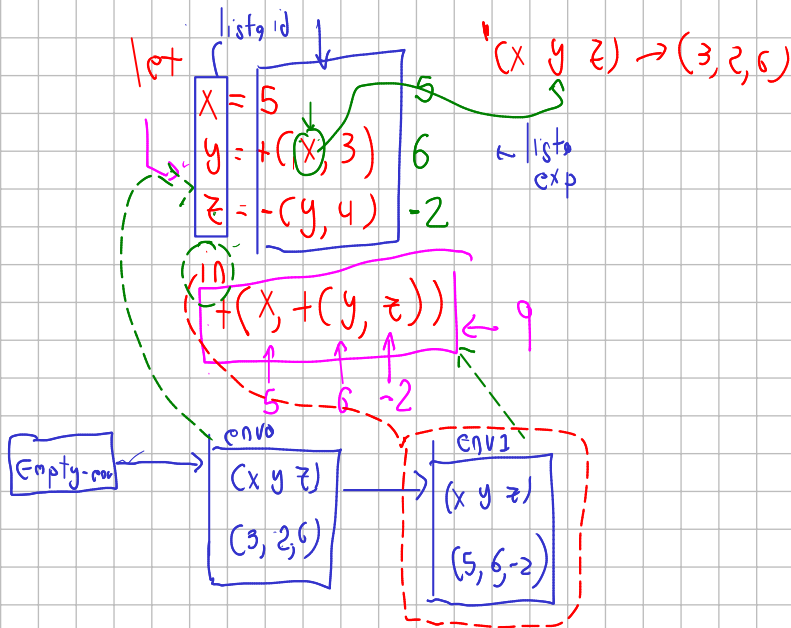
- Primero se evalúa la subexpresión $-(x,4)$. Dado que x vale 4 en el ambiente en el que se evalúa la expresión, el valor de la expresión es 0.
- Como el valor de la subexpresión $-(x,4)$ es 0, se evalúa la subexpresión $*(y,10)$.
- Finalmente, el valor de toda la expresión `if` es 20.

Contenido

- 1 Un interpretador más complejo
- 2 Evaluación de expresiones condicionales
- 3 Ligadura local**

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 ($\text{let-exp } ids \text{ } exprs \text{ } body$) es necesario evaluar las partes derechas de las declaraciones (correspondientes a las expresiones $exprs$) 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 $exprs$).
- 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:

```
let
  x = -(y, 1)
in
  let
    x = +(x, 2)
  in
    add1(x)
```

Semántica de la ligadura local

Ejemplo

- Primero se evalúa 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.
- Posteriormente se evalúa la expresión:

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

en el ambiente env_1 .

Semántica de la ligadura local

Ejemplo

- Primero se evalúa 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.
- Posteriormente se evalúa la expresión:

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

en el ambiente env_1 .

Semántica de la ligadura local

Ejemplo

- Primero se evalúa 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.
- Posteriormente se evalúa la expresión:

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

en el ambiente env_1 .

Semántica de la ligadura local

Ejemplo

- Se evalúa 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.
- Posteriormente 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

- Se evalúa 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.
- Posteriormente 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

- Se evalúa 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.
- Posteriormente se evalúa la expresión $add1(x)$ en el ambiente env_2 .
- Finalmente, el valor de la expresión original es 4.

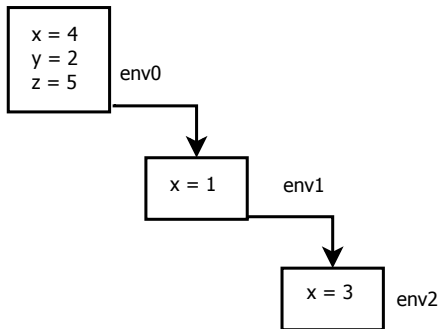
Semántica de la ligadura local

Ejemplo

- Se evalúa 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.
- Posteriormente 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

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



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

```
(define eval-expression
  (lambda (exp env)
    (cases expression exp
      (lit-exp (datum) datum)
      (var-exp (id) (apply-env env id))
      (primapp-exp (prim rands)
                    (let ((args (eval-rands rands env)))
                      (apply-primitive prim args)))
      (if-exp (test-exp true-exp false-exp)
              (if (true-value? (eval-expression test-exp env))
                  (eval-expression true-exp env)
                  (eval-expression false-exp env)))
      (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)))
```

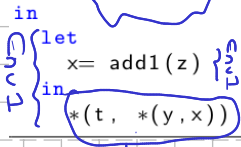
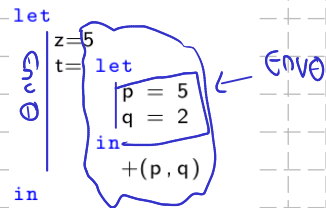
Semántica de la ligadura local

Ejemplo

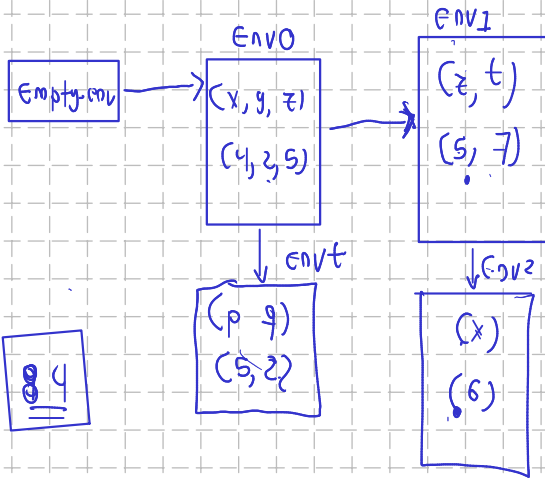
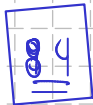
Sea el ambiente env_0 con símbolos (x y z) y valores (4 2 5) el ambiente inicial de computación. Evaluar:

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

$(x, y, z) \ (4, 2, 5)$



$* (7, * (2, 6)) \rightarrow$



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:

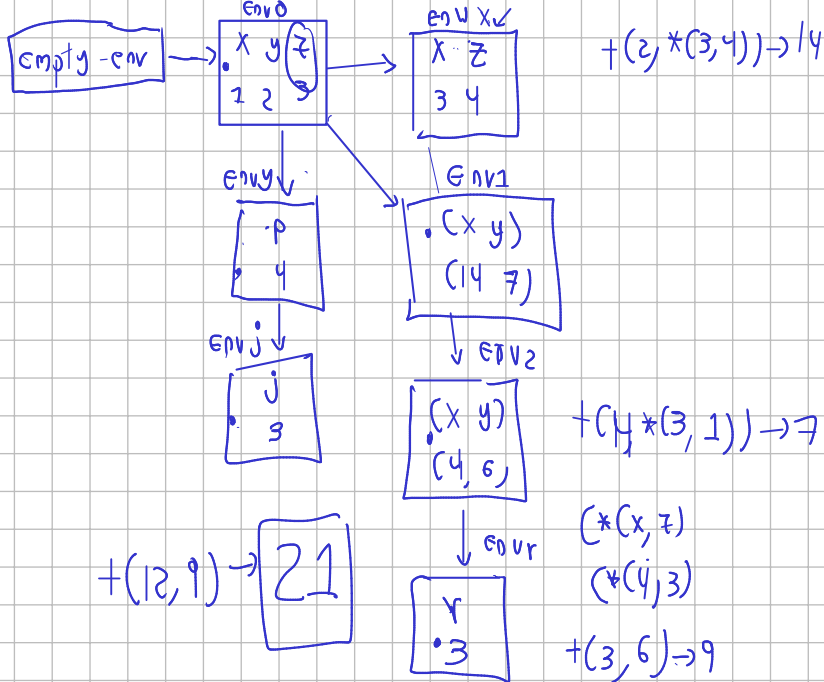
```
let
  y=5
  m= let
    t = sub1(y)
  in
    *(t, x)
in
  let
    y= if sub1(y) then +(add1(y), m) else sub1(+(y, m))
  in
    let
      t = -(y, m)
    in
      +(t, +(y, -(z, 3)))
```

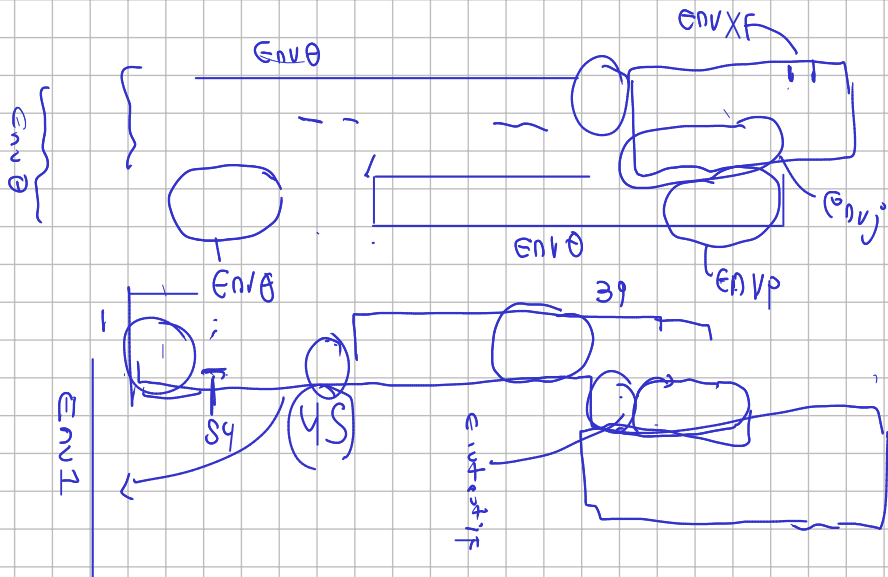
Dibuje los ambientes creados en la evaluación de la expresión.

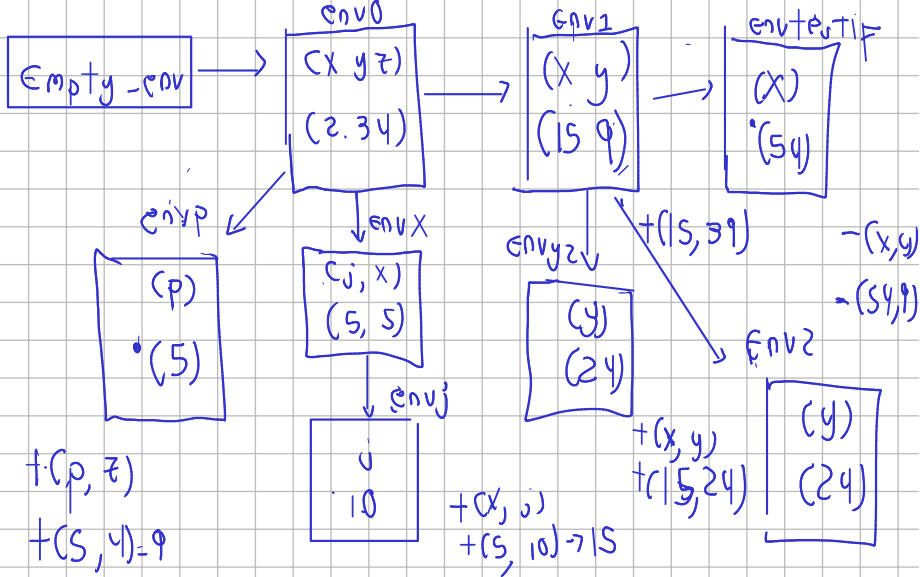
Env0 (x,y,z) -> (1,2,3)

let
 3 0 [x = let x = 3 z = 4 in + (y, *(x,z)) . Env0
 0 (y) = let p = 4 in let j = 3 in + (p, *(j,x)) Env0 Env1
 in Env1

let
 2 1 x = 4 y = 6
 2 in
 3 2 + (*(x,z), let r = 3 in + (r,y))
 Env2 Env1







Preguntas

?

- Semántica de la creación y aplicación de procedimientos.