Laboratorio de Lenguajes de Programación

Evaluación perezosa Parte II

Manuel Soto Romero

Universidad Nacional Autónoma de México
Facultad de Ciencias

12 de octubre de 2016

Objetivos

- Recordar en qué consiste la evaluación perezosa.
- Definir cerraduras de expresiones y resaltar su utilidad.
- Definir el concepto de punto estricto.
- Implementar un intérprete perezoso para `FAE` usando cerraduras de expresiones y puntos estrictos.

A considerar...

¿Qué consideraciones debemos tener al implementar comportamiento perezoso?

- No almacenamos las variables con su valor en el ambiente.
- Ahora guardaremos las variables con una expresión sin evaluar.

Cerraduras de expresiones

Para guardar las expresiones sin evaluar usaremos cerraduras de expresiones. Haremos algo muy similar a las cerraduras de funciones.

Las cerraduras de expresiones almacenarán:

- Las expresión propiamente dicha.
- El ambiente donde fue definida.

¿Por qué es necesario guardar el ambiente?

Ejemplo

Veamos cómo se evaluaría una expresión si sólo guardamos la variable con la expresión sin guardar el ambiente donde fue definida.

Tenemos el siguiente código:

```
{with {a {+ 4 5}}
   {with {b {+ a a}}
      {with {c b}
         {with {a 4}
            c}}}}
```

¿Cómo queda el ambiente variable-expresión para este código?

a	4
c	b
b	{+ a a}
a	{+ 4 5}

Ejemplo

¿Cómo se evalúa c?

1. Buscamos `c` en el ambiente desde el tope. Obtenemos `b`.
2. Buscamos `b` en el ambiente desde el tope. Obtenemos `{+ a a}`.
3. Buscamos evaluar `{+ a a}`. Bucamos `a` en el ambiente desde el tope.

```
{+ a a} = {+ 4 4} = 8
```

Éste no es el resultado esperado

Ejemplo

Veamos cómo se evaluaría la misma expresión si guardamos la variable con la expresión y el ambiente donde fue definida.

1. Introducimos `a` con `{+ 4 5}` y hacemos referencia al ambiente vacio pues ahí fue definida la expresión.

a	{+ 4 5}	vacio

env1

2. Introducimos `b` con `{+ a a}` y hacemos referencia al ambiente anterior pues ahí fue definida la expresión.

b	{+ a a}	env1

env2

Ejemplos

3. Introducimos `c` con `b` y hacemos referencia al ambiente anterior pues ahí fue definida la expresión.

c	b	env2

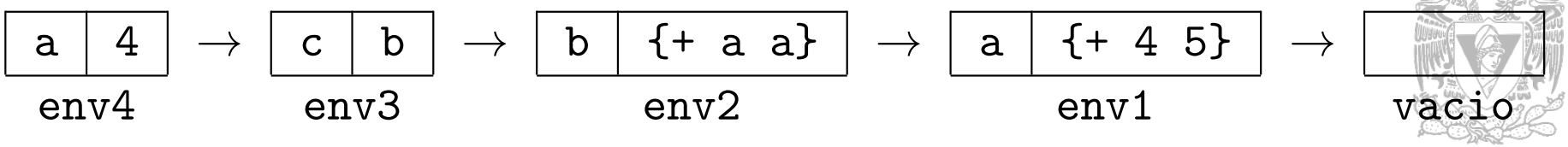
env3

4. Introducimos `a` con `4 5` y hacemos referencia al ambiente anterior pues ahí fue definida la expresión.

a	4	env3

env4

El ambiente final es una cadena de ambientes



Ejemplo

Dado que la expresión `c` que queremos evaluar está en el cuerpo del último `with`, nos encontramos en el ambiente `env4`:

a	4	env3

`env4`

5. Buscamos el valor de `c` en el ambiente `env3`:

c	b	env2

`env3`

6. Buscamos el valor de `b` en el ambiente `env2`:

b	{+ a a}	env1

`env2`

Ejemplo

7. Para aplicar la suma buscamos el valor de a en el ambiente `env1`:

a	{+ 4 5}	vacio

`env1`

Obtenemos finalmente

```
{+ {+4 5} {+ 4 5}}
```

Este sí es el valor esperado

Representando cerraduras de expresiones

Para representar las cerraduras de expresiones, ampliaremos el tipo de dato: `FAE/L-Value`:

```
(define-type FAE/L-Value
   [numV (n number?)]
   [closureV (param symbol?) (env Env?)]
   [exprV (expr FAE/L?) (env Env?)])
```

Puntos estrictos

En el ejemplo anterior, terminamos con la expresión:

{+ {+ 4 5} {+ 4 5}}

Debemos poder evaluar esa expresión, pues el resultado de la suma no debe postergarse, debemos forzar su evaluación.

En algunos casos (no sólo en la suma) debemos forzar la evaluación de algunas expresiones para seguir con la interpretación.

"A los puntos donde la implementación de un lenguaje perezoso fuerza la reducción de una expresión a un valor (si existe) se les llama puntos estrictos del lenguaje"
(Krishnamurthi, 2003)

Condicional 0

Para ilustrar el concepto de punto estricto. Agregaremos otra primitiva a nuestro lenguaje.

Esta primitiva revisa si una condición se evalúa a 0 en cuyo caso regresa un valor y en caso contrario otro:

```
{if0 {+ 2 -2} 17 29}
```

```
(define-type FWAE/L
   [numS (n number?)]
   [binopS (f procedure?) (l FWAE/L?) (r FWAE/L?)]
   [withS (name symbol?) (named-expr FWAE/L?) (body FWAE/L?)]
   [idS (name symbol?)]
   [if0S (expr FWAE/L?) (then-expr FWAE/L?) (else-expr FWAE/L?)]
   [funS (param symbol?) (body FWAE/L?)]
   [appS (fun-expr FWAE/L?) (arg-expr FWAE/L?)])
```

`FAE` es similar pero omitimos `with`.

Puntos estrictos en FAE/L

¿Qué puntos estrictos encontramos en el lenguaje FAE/L?

- Operaciones binarias.
- Condicional de `if0`.
- La primera expresión de la aplicación de funciones.

Análisis sintáctico

La evaluación perezosa no influye en el análisis sintáctico. Todo se queda igual. Sólo hay que agregar el caso para `if0`:

`parse`

```
(if0S (parse (cadr sexp)) (parse (caddr sexp)) (parse (cadddr sexp)))
```

`desugar`

```
(if0 (desugar expr) (desugar then-expr) (desugar else-expr))
```

¿Cómo forzamos la evaluación de expresiones?

Necesitamos una función que aplique el punto estricto:

```
(define (strict e)
   (match e
      [(exprV expr env) (strict (interp expr env))]
      [else e]))
```

La evaluación perezosa consiste entonces en usar el tipo `exprV` para postergar la evaluación y usar `strict` cuando sea necesario aplicar la evaluación.

Interpretación de números

`(num n)` $\rightarrow$ `(numV n)`

Interpretación de identificadores

`(id v)` $\rightarrow$ `(lookup v env)`

Interpretación de operaciones binarias

```
(binop f l r) →
(binopf f (interp l env) (interp l env))
```

Aquí hay un punto estricto

```
(define (binopf l r)
   (numV (f (numV-n (strict l)) (numV-n (strict r)))))
```

Interpretación del condicional 0

```
(if0 expr then-expr else-expr) →
(if (interp expr env)
   (interp then-expr env)
   (interp else-expr env))
```

Aquí hay un punto estricto

No basta con interpretar la condición del if, hay que forzar la evaluación y determinar si el resultado es cero.

```
(define (num-zero? n)
   (zero? (numV-n (strict n))))
```

```
(if0 expr then-expr else-expr) →
(if (num-zero? (interp expr env))
   (interp then-expr env)
   (interp else-expr env))
```

Interpretación de funciones

```
(fun bound-id bound-body env) →
(closureV bound-id bound-body env)
```

Interpretación de aplicación de funciones

```
(app fun-expr arg-expr) →
(let ([fun-val (interp fun expr env)])
   (interp (closureV-body fun-val)
      (aSub (closureV-param fun-val)
         (interp arg-expr env)
         (closureV-env fun-val))))
```

Aquí hay un punto estricto

Hay que considerar dos cosas:

- La primera expresión de la aplicación de función es un punto estricto.
- El argumento de la aplicación debe ser postergado, necesitamos una cerradura de expresión.