Laboratorio de Lenguajes de Programación

Recursión - Parte II

Manuel Soto Romero

Universidad Nacional Autónoma de México
Facultad de Ciencias

26 de octubre de 2016

Objetivos

- Recordar cómo se definen funciones con nombres.
- Definir una función recursiva usando asignaciones locales.
- Presentar una estrategia de tres pasos para implementar recursión.
- Construir un intérprete recursivo.

Volviendo en el tiempo...

Al inicio del semestre vimos que las funciones con nombres eran azúcar sintáctica de asignaciones locales. Por ejemplo:

```
(define (area-triangulo b h)
   (/ (* b h) 2))
```

Se ve cómo:

```
(let ([area-triangulo (lambda (b h) (/ (* b h) 2))])
    (area-triangulo 20 27))
```

En nuestro lenguaje objetivo:

```
{with {area-triangulo {fun {b h} {/ {* b h} 2}}}
    {area-triangulo 20 27}}
```

Definiendo funciones recursivas

¿Cómo se ve la siguiente función?

```
(define (fact n)
  (if (zero? n))
     1
     (* n (fact (sub1 n)))))

(let ([fact
       (λ (n) (if (zero? n) 1 (* n (fact (sub1 n)))))])
  (fact 5))

{with {fact
        {fun {n} {if0 n 1 {* n {fact {- n 1}}}}}}}
  {fact 5}}
```

Evaluando funciones recursivas

Ambiente

n	5
fact	(closureV 'n (if0 'n ...) (mtSub))

- Lo primero es meter a `fact` en el ambiente con la cerradura de la función correspondiente. **Esta cerradura tiene a `mtSub` como ambiente**.
- Ahora queremos evaluar `{fact 5}`, por lo que metemos `n` con 5 al ambiente y buscamos a `fact` en el ambiente. **Esto nos regresa una cerradura**.
- Tenemos que evaluar el cuerpo de la función `(if0 5 1 (* 5 (fact 4)))` en el ambiente de la cerradura.

¡Pero está vacía!

¿Qué hacemos entonces?

¿Cómo se evalúan las funciones recursivas entonces?

El código anterior aunque es sintácticamente correcto, no lo es semánticamente pues el identificador fact interior queda libre.

Necesitamos una manera de hacer referencia al mismo identificador. Podemos usar la siguiente estrategia de tres pasos apoyándonos de la estructura de dato «caja»:

1. Nombrar una caja vacía.
2. Mutar la caja para que su contenido sea él mismo.
3. Para obtenerlo, usar el nombre definido previamente.

Ejemplo

Nombramos una caja vacía (con 1729 por convención)

```
(let ([fact (box 1729)])
   ...
```

Mutamos la caja para que su contenido sea él mismo

```
(let ([fact (box 1729)])
   (begin
      (set-box! fact
         (λ (n)
            (if (zero? n) 1 (* n ((unbox fact) (sub1 n))))))
   ...)
```

Para obtenerlo usamos el nombre definido previamente

```
(let ([fact (box 1729)])
   (begin
      (set-box! fact (λ (n) (if ...)))
      ((unbox fact) 5)))
```

¡Esto es lo que hace letrec!

¿Cómo lo implementamos?

Para empezar, agregaremos la primitiva rec a nuestro lenguaje que será el equivalente a letrec. Su sintaxis es parecida a la de un with.

```
{rec {fact
        {fun {n} {if0 n 1 {* n {fact {- n 1}}}}}}
  {fact 5}}
```

Lo que queremos lograr es que al interpretar se aplique la estrategia de tres pasos