Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

•••

Karla Ramírez Pulido

Recursión
Parte I

Recursión

Funciones Recursivas

- Una función se llama dentro del cuerpo de la misma función
- Debe de tener un caso base (termine la recursión)

Ejemplos:

Factorial

Fibonacci

Función factorial

```
factorial 0 = 1

factorial n = (n  *  (factorial (n  -  1))
```

Ejemplo:

factorial 0 = 1

factorial 1 = (1 * (factorial 0)) = (1 * 1) = 1

factorial 2 = (2 * (factorial 1)) = (2 * (1 * (factorial 0)))

= (2 * (1 * 1)) = 2 * 1 = 2

Sigamos...

factorial 1,000,000 = (1,000,000 * (factorial 999,999))

= (1,000,000 * (999,999 * (factorial 999,998)))

...

= (1,000,000 * (999,999 * ... (factorial 0)))

= (1,000,000 * (999,999 * ... (1 * 1))

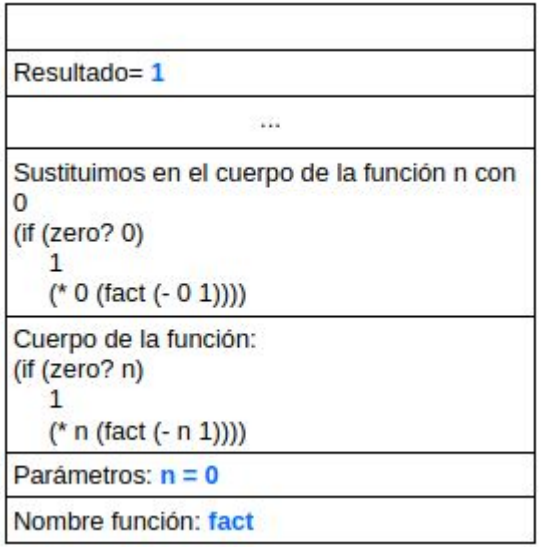
¿Cuántas llamadas recursivas a factorial se hicieron?

¿Cuántas operaciones quedaron pendientes de realizarse antes de llegar al caso base?

Veamos el stack

Un registro de activación:

- nombre de la función
- parámetros de la función con sus respectivos valores (parám reales)
- cuerpo de la función
- variables utilizadas
- ...
- resultado de la función



Registros de activación

factorial 3

Resultado= (* 2 (fact 1))
...
Sustituimos en el cuerpo de la función n con 2 (if (zero? 2) 1 (* 2 (fact (- 2 1))))
Cuerpo de la función: (if (zero? n) 1 (* n (fact (- n 1))))
Parámetros: n = 2
Nombre función: fact
Resultado= (* 3 (fact 2))
...
Sustituimos en el cuerpo de la función n con 3 (if (zero? 3) 1 (* 3 (fact (- 3 1))))
Cuerpo de la función: (if (zero? n) 1 (* n (fact (- n 1))))
Parámetros: n = 3
Nombre función: fact

Registros de activación

factorial 3

Resultado= 1
...
Sustituimos en el cuerpo de la función n con 0 (if (zero? 0) 1 (* 0 (fact (- 0 1))))
Cuerpo de la función: (if (zero? n) 1 (* n (fact (- n 1))))
Parámetros: n = 0
Nombre función: fact
Resultado= (* 1 (fact 0))
...
Sustituimos en el cuerpo de la función n con 1 (if (zero? 3) 1 (* 1 (fact (- 1 1))))
Cuerpo de la función: (if (zero? n) 1 (* n (fact (- n 1))))
Parámetros: n = 1
Nombre función: fact
Resultado= (* 2 (fact 1))

¿Cuántos registros de activación se crean en tiempo de ejecución con:

factorial de 0 = 1

factorial de 3 = 4

factorial de 1,000,000 = 1,000,001

Para llegar a la implementación en nuestro intérprete, primero:

- Añadir condicional: if0
- Añadir funciones recursivas: rec

Gramática

Añadir condicional if0

if0 expr-cond expr-then expr-else

<CFAE> ::= <num>

| {+ <CFAE> <CFAE>}

| {* <CFAE> <CFAE>}

| <id>

| {fun {<id>} <CFAE>}

| {<CFAE> <CFAE>}

| {if0 <CFAE> <CFAE> <CFAE>}

Constructor: if0

if0 evalúa la expr-condicional, si ésta es 0 entonces regresa la rama de la expr-then, sino la rama de la expr-else

{if0 {+ 5 -5}

1

2}

Evalúa a 1

Gramática

Añadimos expresiones recursivas

rec {id-fun def-fun body-rec}

<RCFAE> ::= <num>

| {+ <RCFAE> <RCFAE>}

| {* <RCFAE> <RCFAE>}

| <id>

| {fun {<id>} <RCFAE>}

| {<RCFAE> <RCFAE>}

| {if0 <RCFAE> <RCFAE> <RCFAE>}

| {rec {<id> <RCFAE>} <RCFAE>}

Función factorial

id = fac

def-fun = {fun {n}{if0 n
1
{* n {fac {- n 1}}}}

body-rec = {fac 5}

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

Función factorial

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

{fac 5}

{{fun(n) {if0 n 1 {* n {fac (- n 1)}}}} 5}

{fun(5) {if0 5 1 {* 5 {fac (- 5 1)}}}}

Eval body = {if0 5 1 {* 5 {fac 4}}}

= {* 5 {fac 4}}

= {* 5

{{fun(n) {if0 n 1 {* n {fac (- n 1)}}}} 4}}

= { * 5

{fun(4) {if0 4 1 {* 4 {fac (- 4 1)}}}} }

Función factorial

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

= { * 5

{if0 4 1 {* 4 {fac 3}}} }

= {* 5 {* 4 {fac 3}}}

= {* 5 {* 4 {* 3 { fac 2}}}}

= {* 5 {* 4 {* 3 {* 2 {fac 1}}}}}

= {* 5 {* 4 {* 3 {* 2 {* 1 {fac 0}}}}}}

Función factorial

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

= {* 5 {* 4 {* 3 {* 2 {* 1 1}}}}}

= {* 5 {* 4 {* 3 {* 2 1}}}}

= {* 5 {* 4 {* 3 2}}}

= {* 5 {* 4 6}}

= {* 5 24}

= 120

fac 5

¿Cuántos registros de activación se crearon en ejecución?

¿Cuándo empezaron a liberar espacio en memoria dichos registros de activación?

MUY IMPORTANTE: quedaron al menos 6 registros de activación en memoria PENDIENTES por resolver...

Puntos fijos

Si se cumple que f(x) = x

Ejemplos:

- La función f(x) = 0 tiene exactamente un punto fijo porque f(n)= n solo cuando n=0.
- f(x) = x^2

  tiene 2 puntos fijos cuando

  x= 0 y x= 1

- f(x) = x

  tiene una infinidad de puntos fijos

Objetos recursivos y cíclicos

Un objeto cíclico no contiene sólo referencias a objetos de su mismo tipo, sino contiene referencias a él mismo.

Ejemplos:

1. Árbol binario
2. Gráfica
3. Páginas web

¿Cuál es cíclica y cuál recursiva?

Diferencias entre with y rec

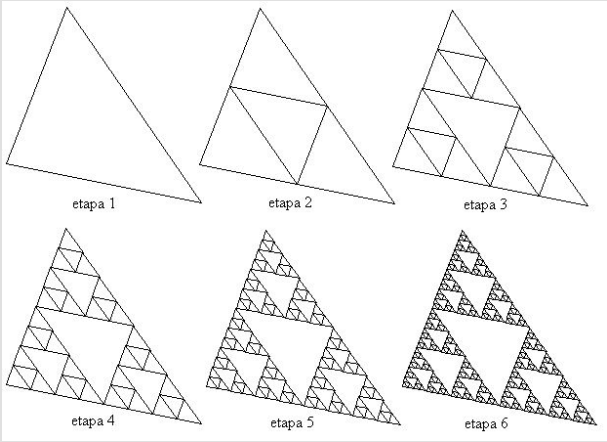
with y rec tienen la misma estructura en principio pero se usan de manera distinta

Si lo que quiero es expresar una asignación de id a un valor que NO es una función recursiva uso with y no rec.

{rec {n 5} {+ n 10}} NO

{with {n 5} {+ n 10}} SÍ

Implementación de funciones recursivas



Triángulo de Sierpinsky

Definimos tipos de datos

Valores

```
(define-type RCFAE-Value
        [numV (n number?)]
        [closureV  (param symbol?)
                   (body RCFAE?)
                   (env Env?)])
```

Definimos tipos de datos

Ambientes

```
(define-type Env
    [mtSub]
    [aSub(name symbol?)
        (value RCFAE-Value?)
        (env Env?)]
    [aRecSub (name symbol?)
             (value boxed-RCFAE-Value?)
             (env Env?)])
```

Introduciendo CAJAS

Cajas: estructuras de datos que pueden almacenar un único elemento

- box
- unbox
- box?

```
(define (boxed-RCFAE-Value? v)
        (and   (box? v)
               (RCFAE-Value? (unbox v))))
```

Función lookup

Busca un símbolo en el ambiente

;; lookup : symbol env→RCFAE-Value

```
(define (lookup name env)
   (type-case Env env
      [mtSub() (error'lookup "no binding for
      identifier")]
      [aSub(bound-name bound-value rest-env)
         (if (symbol=? bound-name name)
             bound-value
             (lookup name rest-env))]
      [aRecSub(bound-name boxed-bound-value rest-env)
          (if (symbol=? bound-name name)
              (unbox boxed-bound-value)
              (lookup name rest-env))]))
```

```
;; cyclically-bind-and-interp : symbolRCFAEenv→env


(define (cyclically-bind-and-interp bound-id  named-expr  env)
    (local ([define value-holder (box (numV 1729))]
                    [define  new-env (aRecSub  bound-id  value-holder  env)]
                    [define  named-expr-val (interp   named-expr   new-env)])
                        (begin
                                (set-box!  value-holder  named-expr-val)
                                new-env)))
```

Recursión: Intérprete (1)

```
;; interp : RCFAE env→RCFAE-Value

(define (interp expr env)

    (type-case RCFAE expr

        [num (n) (numV n)]

        [add (l r) (num+ (interp l env) (interp r env))]

        [mult (l r) (num*(interp l env) (interp r env))]

        [if0 (test-expr  then-expr  else-expr)

                (if (num-zero? (interp test-expr env))

                        (interp then-expr env)

                        (interp else-expr env))]
```

Recursión: Intérprete (2)

;; interp : RCFAE env→RCFAE-Value

```
[id (v) (lookup v env)]

[fun (bound-id bound-body)

     (closureV bound-id bound-body env)]

[app (fun-expr  arg-expr)

     (local ([define fun-val (interp fun-expr env)])

          (interp (closureV-body fun-val) (aSub (closureV-param fun-val)

                                                (interp arg-expr env)

                                                (closureV-env fun-val))))]
```

Recursión: Intérprete (3)

;; interp : RCFAE env→RCFAE-Value

```
[rec (bound-id named-expr bound-body)

     (interp bound-body

          (cyclically-bind-and-interp bound-id  named-expr  env))]))
```

Ejecución de interp: números y sumas

> (interp (numV 3) ())

(numV 3)

> (interp (add (numV 3) (numV 2)) ())

Sumar el interp del lado izq y el interp del lado der.

(num+ (interp (numV 3) ()) (interp (numV 2) ()))

(+ 3 2) => 5 => (numV 5)

```
(define (interp expr env)
  (type-case RCFAE expr
    [num (n) (numV n)]
    [add (l r) (num+ (interp l env)
                     (interp r env))]
```

Ejecución de interp: ids y funciones

> (interp (id z) ( ))

(lookup (id z) ( ))

”no binding for identifier”

```
(define (interp expr env)

  (type-case RCFAE expr

                …

   [id (v) (lookup v env)]  …))
```

```
(define (lookup name env)

  (type-case Env env

    [mtSub() (error’lookup ”no binding for
    identifier”)]

    [aSub(bound-name bound-value rest-env)

       (if (symbol=? bound-name name) bound-value

          (lookup name rest-env))]
```

```
[aRecSub(bound-name boxed-bound-value
rest-env)

   (if (symbol=? bound-name name)

         (unbox boxed-bound-value)

         (lookup name rest-env))]))
```

Ejecución de interp: funciones

```
(define (interp expr env)
  (type-case RCFAE expr
    …
    [fun (bound-id bound-body)
         (closureV bound-id
                   bound-body
                   env)]
```

```
> (interp (fun (id x) (id x)) ( ) )
  (closureV (id x) (id x) ( ) )
```

Ejecución de interp: if0

> (interp (if0 (num 5) (num 1) (num 2)) ( ))

(**if** (num-zero? (interp (num5) ( )))

(num-zero? (num 5)) NOOO

(**if** false

interpreto la expresión else-expr

i.e. (interp (num 2) ( ))

= (num 2)

```
(define (interp expr env)

…

 [if0 (test-expr  then-expr  else-expr)

      (if (num-zero? (interp test-expr env))

                     (interp then-expr env)

                     (interp else-expr env))]
```

Gracias