Universidad Nacional Autónoma de México
Facultad de Ciencias

Lenguajes de Programación
Karla Ramírez Pulido

Implementación de Funciones con y sin ambientes

IMPLEMENTACIóN DE FUNCIONES

Definición de función

Aplicaciones de función

Extendiendo el lenguaje WAE a FWAE

```
<FWAE> ::= <num>
           |  {+  <FWAE> <FWAE>}
           | {with {<id> <FWAE>} <FWAE>}
           | <id>
           | {fun  {<id>}  <FWAE>}
           | {<FWAE>  <FWAE>}
```

Constructor

```
(define-type  FWAE

      [num (n  number?)]

      [add  (lhs   FWAE?) (rhs   FWAE?)]

      [with  (name  symbol?) (named-expr   FWAE?) (body FWAE?)]

      [id (name   symbol?)]

      [fun (param   symbol?) (body  FWAE?)]

      [app  (fun-expr   FWAE?)  (arg-expr   FWAE?)])
```

Definición de la función interp (sin ambientes)

```
(define (interp  expr)
    (type-case FWAE expr
        [num (n) expr]
        [add (l  r) (add-numbers  (interp  l) (interp  r))]
        [with (bound-id  named-expr  bound-body)
              (interp  (subst  bound-body  bound-id  (interp named-expr)))]
        [id (v) (error ’interp ”free identifier”)]
```

Continuación de interp (sin ambientes)

```
[fun (bound-id  bound-body)  expr]

[app (fun-expr  arg-expr)

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

          (interp  (subst  (fun-body  fun-val)

                           (fun-param  fun-val)

                           (interp  arg-expr))))]  ))
```

Ejecuciones de interp:

```
> (interp ‘(num 4))

(num 4)

> (interp ‘(id x))

“free identifier”

> (add (num 3) (num 2))

(add-numbers (interp (num 3))  (interp (num 2)))
```

```
(define (interp   expr)
    (type-case FWAE expr
        [num (n) expr]
        [add (l  r) (add-numbers
                        (interp  l) (interp r))]
        [id (v) (error ’interp
                        ”free  identifier”)]
```

Ejecuciones de interp:

```
> (interp ‘(num 4))

(num 4)

> (interp ‘(id x))

“free identifier”

> (interp  ‘(add (num 3) (num 2)) )

(add-numbers (interp (num 3))  (interp (num 2)))
```

Casos de interp del lado izq y der de la suma:

```
> (interp (num   3))

(num   3)


> (interp (num   2))

(num   2)
```

Ejecuciones de interp:

```
(add-numbers  (interp (num 3))  (interp (num 2)))

(add-numbers        (num 3)           (num 2))

(add-numbers  3   2)

;;Función auxiliar: add-numbers recibe dos instancias en WAE específicamente

 ;;de (num n) y obtiene el 2nd. de esa lista para sumarlo con la función + de
;;Racket y luego vuelve a ponerle la etiqueta num al resultado  i.e evalúa 3 + 2 = 5

⇒  (num 5)
```

Ejecuciones de interp para FUNCIONES:

```
> (interp ‘{fun {n}  n} )

  {fun {n}  n}

> (interp ‘{fun {x} {+ x 3}} )

   {fun {x} {+ x 3}}

> (interp ‘{fun{x} {-  x  a}})

   {fun {x} {- x  a}}
```

```
(define (interp   expr)

       (type-case FWAE expr

      …

      [fun (bound-id   bound-body)   expr]
```

Ejecuciones de interp:

```
> (interp '{with  {x   {+ 1  2}}  x} )

 [with   ( x   {+ 1  2}   x)

          bound-id   named-expr  bound-body

       (interp (subst   x   x (interp {+ 1  2})))]
```

Tenemos una llamada interna

```
(interp {+ 1  2})
```

y otra de subst

```
(define (interp   expr)

   (type-case FWAE expr

      [num (n) expr]

      [add (l  r)

       (add-numbers  (interp   l) (interp   r))]

      [with (bound-id   named-expr  bound-body)

       (interp  (subst   bound-body  bound-id
      (interp named-expr)))]

      [id (v) (error 'interp "free identifier")]
```

Ejecuciones de interp:

(interp (subst x x (interp {+ 1 2})))

Tenemos una llamada interna

(interp {+ 1 2})

⇒ (add-numbers (interp 1) (interp 2))

(add-numbers 1 2) ⇒

(add-numbers 3) ⇒ (num 3)

Resultado de ese interp es: (num 3)

```
(define (interp expr)
  (type-case FWAE expr
    [num (n) expr]
    [add (l r)
     (add-numbers (interp l) (interp r))]
    [with (bound-id named-expr bound-body)
     (interp (subst bound-body bound-id
    (interp named-expr)))]
    [id (v) (error 'interp "free identifier")]
```

Ejecuciones de interp:

(interp (subst x x (num 3)))

Ahora hacemos la llamada de subst

(subst x x (num 3))

⇒ (num 3)

Entonces:

(interp (subst x x (num 3)))

= (interp (num 3))

Solo para recordar qué hace subst:

;;subst: sustituye var por val en expr

(subst expr var val)

Ejemplos:

1. (subst x y 0)

   ⇒ x

2. (subst (+ y 1) y 0)

   ⇒ (+ 0 1)

Ejecuciones de interp:

Por último: (interp (num 3))

= (num 3)

Y teníamos en un principio

```
(interp '{with {x {+ 1 2}} x} )
```

⇒ x = 3 i.e. (num 3)

Ejecuciones de interp:

```
> (interp   {with {foo  {fun {y}  y}}
                        {foo  3}} )
```

```
[with (bound-id   named-expr  bound-body)
   (interp  (subst   bound-body  bound-id
                             (interp named-expr)))]
```

```
[with  (foo   {fun {y}   y}   {foo  3})
= (interp (subst  {foo  3}  foo  (interp {fun {y} y}))) ]
= (interp (subst  {foo  3}   foo   {fun {y} y} ))
```

Ejecuciones de interp:

```
= (interp (subst  {foo  3}   foo   {fun {y} y} ))
```

```
(subst {foo  3}   foo   {fun {y} y} )

         expr        var       val

       = { {fun {y}  y}    3 }
```

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

   (interp  (subst   bound-body  bound-id

                                (interp named-expr)))]
```

;; Aplicación de función { <FWAE> <FWAE> }

Ejecuciones de interp:

```
{ {fun {y}  y}    3 }

  fun-expr     arg-expr

⇒  [app ( {fun {y}  y}    3 ) ]

fun-val  =  (interp  {fun {y}  y})

         =  {fun {y}  y}
```

```
[fun (bound-id   bound-body)   expr]

[app (fun-expr   arg-expr)

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

       (interp

          (subst  (fun-body   fun-val)

                  (fun-param   fun-val)

                  (interp   arg-expr))))]  ))
```

Ejecuciones de interp:

```
⇒  [app ( {fun {y}  y}    3 ) ]

    (interp

          (subst  (fun-body   {fun {y}  y} )

                      (fun-param   {fun {y}  y} )

                      (interp   3))) ]
```

```
[fun (bound-id   bound-body)   expr]

[app (fun-expr   arg-expr)

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

       (interp

          (subst  (fun-body    fun-val)

                   (fun-param   fun-val)

                   (interp   arg-expr))))]  ))
```

Ejecuciones de interp:

```
;;Función selectora del cuerpo de una función

(fun-body   {fun {y}  y} )

⇒  y

;;Función selectora del parámetro formal de una función

(fun-param   {fun {y}  y} )

⇒   y

(interp   3)

⇒ 3
```

Ejecuciones de interp:

```
⇒ [app ( {fun {y}  y}    3 ) ]
    (interp
        (subst   y   y   3)) ]
   ⇒  (interp
         (num 3) )
       =   (num 3)
```

Implementación con ambientes

Creando ambientes

```
(define-type   Env
      [mtSub]
      [aSub (name   symbol?)  (value   FAE?)  (env   Env?)])
```

Implementación de interp con ambientes

```
(define (interp  expr  env)
        (type-case   FAE   expr
              [num (n) expr]
              [add (l   r) (add-numbers (interp   l   env) (interp   r   env))]
              [id (v)  (lookup   v  env)]
               [fun (bound-id    bound-body)  expr]
```

Continuación de interp

```
[app (fun-expr   arg-expr)
        (local ([define   fun-val   (interp  fun-expr   env)])
            (interp (fun-body   fun-val)
                (aSub (fun-param   fun-val) (interp   arg-expr   env)   env)))]))
```

Ejecuciones de interp:

```
> (interp ‘(num 4)  ( ) )

= (num 4)


> (interp ‘(id x)  ( ) )

= (lookup  (id x)  ( ) )

= “free id”
```

```
(define (interp   expr  env)

    (type-case FWAE expr

        [num (n) expr]

        [id (v) (lookup v  env)]

…

;;Función auxiliar: lookup: id env -> val or
error

(define (lookup   id   env)

    (if (equal=?  id    (car …) …)
```

Ejecuciones de interp:

```
> (interp  '(add (num 3) (num 2))  ( ) )


= (add-numbers (interp (num 3) ( ))
               (interp (num 2) ( )))
```

Ejecutemos las llamadas a interp internas:

```
(define (interp  expr  env)
  (type-case   FAE   expr
      [num (n) expr]
      [add (l   r)
        (add-numbers (interp   l   env)
                     (interp   r   env))]
```

Ejecuciones de interp:

```
(add-numbers (interp (num 3) () )
             (interp (num 2) ( ) ))

= (add-numbers   (num 3) (num 2) )

  (add-numbers   3  2) ⇒  (+  3   2) = 5 ⇒ (num 5)


= (num  5)
```

Casos de interp del lado izq y der de la suma:

```
> (interp  (num   3) ( ))

(num   3)


> (interp  (num   2)  ( ) )

(num   2)
```

Ejecuciones de interp:

```
> (interp  '{fun {n}  n}  ( ) )

{fun {n}  n}

> (interp  '{fun {x} {+ x 3}}  ( ) )

 {fun {x} {+ x 3}}

> (interp  '{fun {x} {-  x  a}}  ( ) )

{fun {x} {- x  a}}
```

```
(define (interp   expr  env)

       (type-case FWAE expr

       …

       [fun (bound-id   bound-body)   expr]
```

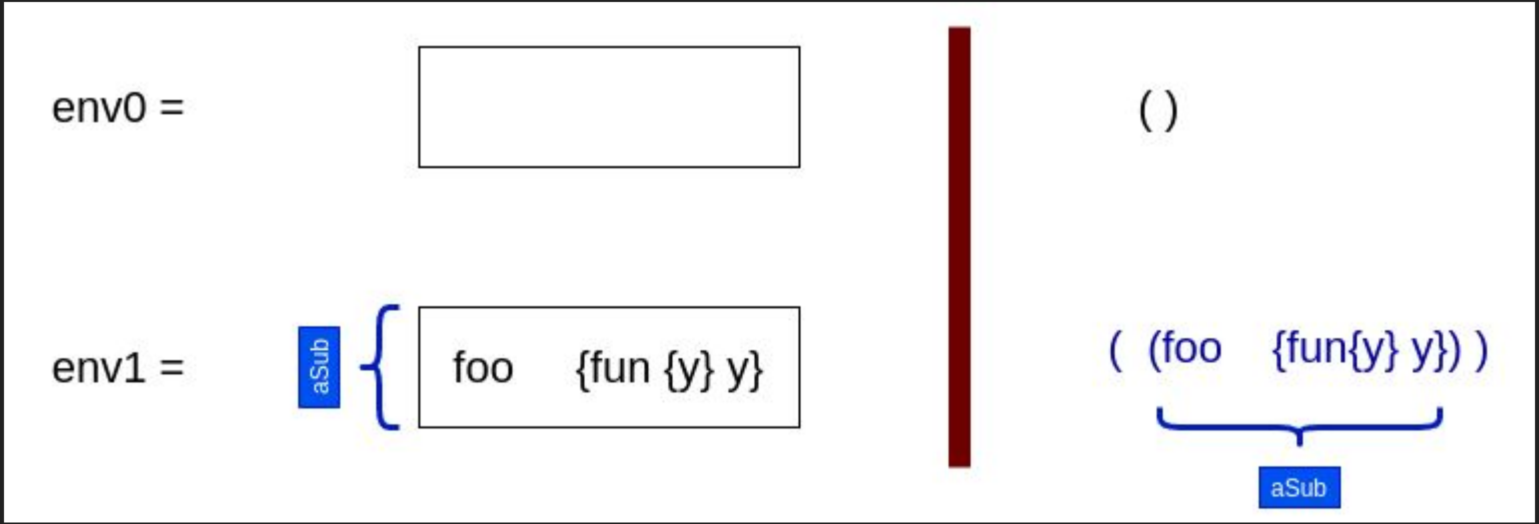
Ejecuciones de interp:

```
> (interp   {with {foo  {fun {y}  y}}
                  {foo  3}}   ( ) )
```

```
[app (fun-expr   arg-expr)
   (local ([define fun-val  (interp fun-expr env)])
      (interp (fun-body   fun-val)
            (aSub (fun-param   fun-val)
            (interp   arg-expr   env)
            env)))]
```

```
[with  (foo   {fun {y}   y}   {foo  3})
= (interp {foo  3}  (aSub  foo  (interp {fun {y} y})  env) ]
                           var       val
```

Ejecuciones de interp con ambientes



(interp {foo 3} **(aSub** foo (interp {fun {y} y}) env)]

i.e. (interp {foo 3} env1)]

Interp de una aplicación de función

```
( interp {foo   3  }    env1 )  ]
   fun-expr   arg-expr


fun-val =  (interp   foo   env1 )
           [lookup  foo  env1]
         = {fun {y}  y}
```

```
[app (fun-expr   arg-expr)
    (local ([define fun-val  (interp fun-expr env)])
        (interp (fun-body   fun-val)
                    (aSub (fun-param   fun-val)
                    (interp   arg-expr   env)   env)))]
```

Interp de una aplicación de función

```
(interp (fun-body {fun {y}  y})
     (aSub  (fun-param {fun {y}  y} )
        (interp   3   env1)   env1)))
```

```
[app (fun-expr   arg-expr)
   (local ([define fun-val  (interp fun-expr env)])
      (interp (fun-body  fun-val )
               (aSub (fun-param   fun-val)
               (interp   arg-expr   env)   env)))]
```

Primero se resuelven las llamadas internas: fun-body, fun-param, e interp de 3 en el ambiente

Interp de una aplicación de función

;;Función selectora del cuerpo de una función:

(fun-body {fun { y } y})

⇒ y

;;Función selectora del cuerpo de una función:

(fun-param {fun { **y** } y})

⇒ y

;;Interpretar un número

(interp 3 env1)

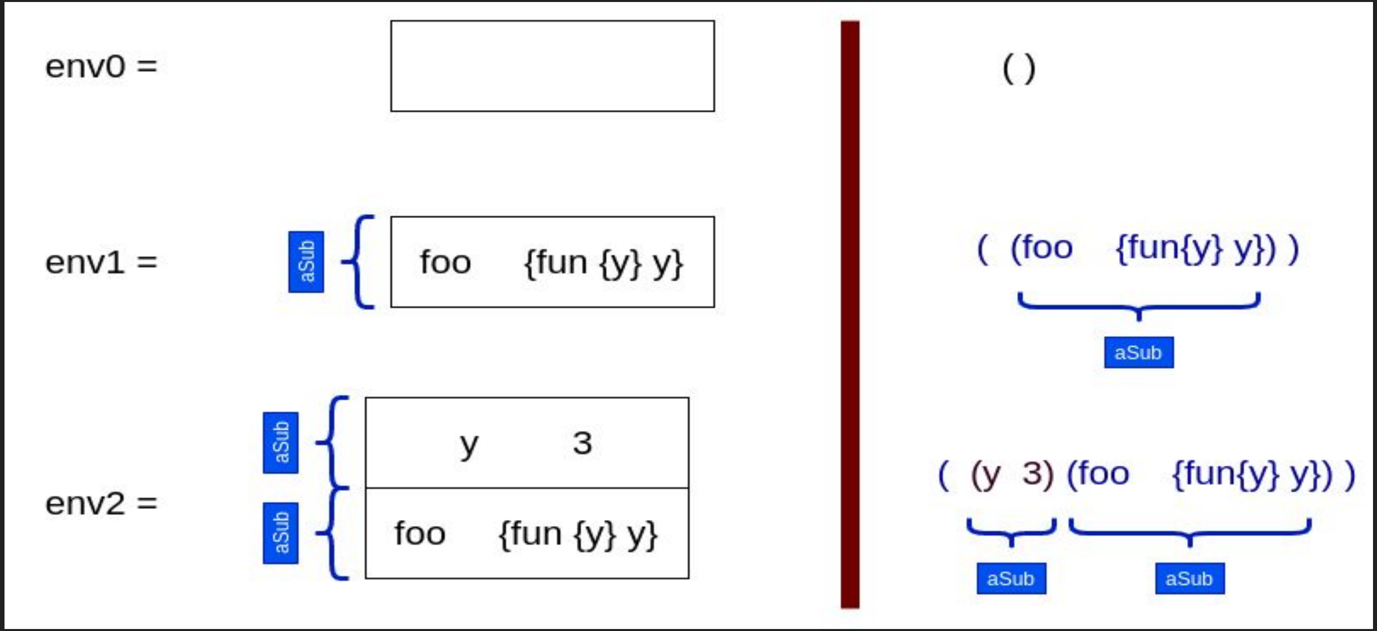
⇒ 3 i.e. (num 3)

Interp de una aplicación de función

```
(interp (fun-body {fun {y}  y})
     (aSub  (fun-param {fun {y}  y} )
         (interp   3   env1)   env1)))
=
(interp  y
     (aSub  y   3  env1))
```

```
[app (fun-expr   arg-expr)
   (local ([define fun-val  (interp fun-expr env)])
       (interp (fun-body  fun-val )
                   (aSub (fun-param   fun-val)
                   (interp   arg-expr   env)   env)))]
```

Interp de una aplicación de función



(interp y

(aSub y 3 env1)) i.e. (interp y env2)

Interp de una aplicación de función

```
( interp   y     env2 )

[id   (y)

   (lookup   y   env2) ]
```

⇒ 3   i.e. (num 3)

Recordemos que veníamos de la expresión:

```
(interp   {with {foo  {fun {y}  y}}

                      {foo  3}}            ( ))
```

i.e.   {foo 3}

{ {fun {y} y}   3}

Asignación [y:= 3]

evaluamos boby-function ⇒   y = 3

Gracias