Universidad Nacional Autónoma de México
Facultad de Ciencias

Lenguajes de Programación

Karla Ramírez Pulido

Cerraduras
(closures)

Cerraduras o Closures

¿Para qué lo usamos?

Implementar alcance estático.

Almacenan la información de una función:

- Parámetros formales.
- Cuerpo de la función.
- Ambiente de dicha función.

Cerradura o *closure* (en inglés)

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

En la función interp

```
[fun (bound-id   bound-body)
      (closureV  bound-id   bound-body   env)]
```

Ambiente

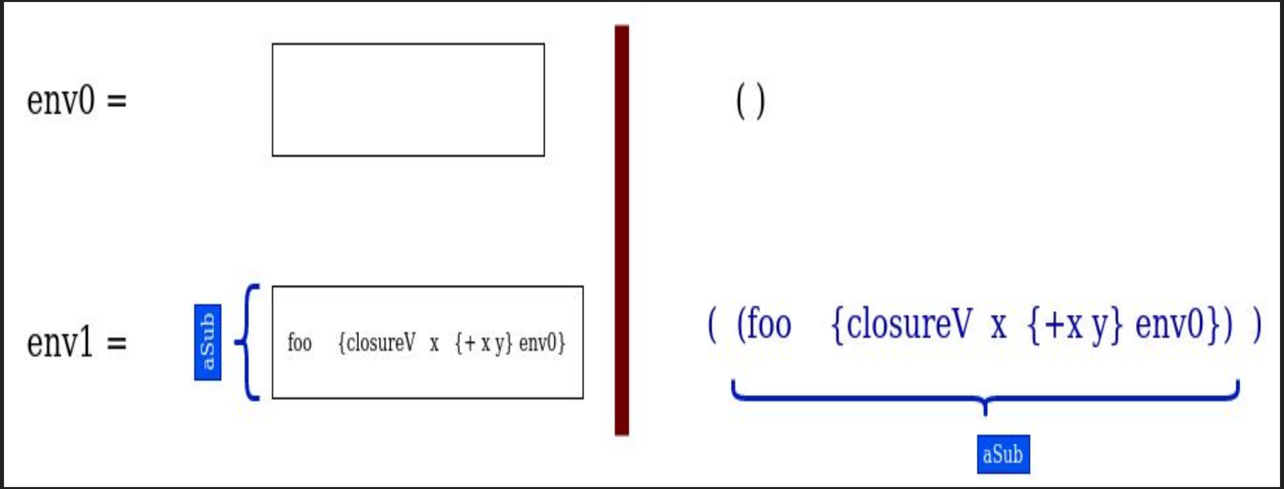
```
(define-type Env
    [mtSub]
    [aSub (name  symbol?) (value  FAE?) (env  Env?)])
```

Ejemplo 1 con cerraduras

(interp ‘{with {foo   {fun {x} {+ x y}}} {+ 2   3}} ( ))

⇒ {+ 2   3}

= 5



env0 =
( )
env1 =
aSub
foo {closureV x {+ x y} env0}
((foo {closureV x {+x y} env0}))
aSub

Definición de la función interp

```
(define (interp  expr  env)
    (type-case FWAE expr
        [num (n) expr]
        [add (l  r) (add-numbers  (interp  l  env) (interp  r  env))]
            ...
        [id (v) (lookup  v   env)]
```

Definición de la función interp

```
[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))))]  ))
```

Ejemplo 2

```
(interp ‘{with {x   3}
              {with {foo  {fun {y} {+  x  y}}}
                 {with   {x    5}
                      {f   4}}}}                  ( )
)
```

expr **env**

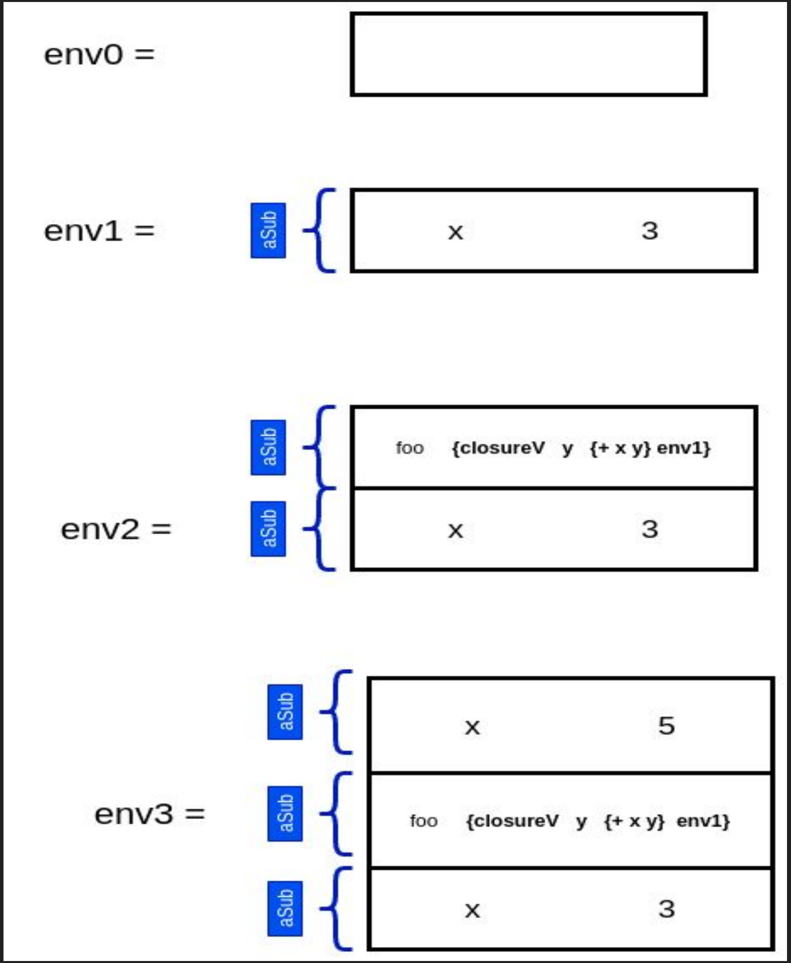
1. Creamos el ambiente
2. Evaluamos el cuerpo del *with* es decir:

   (f   4)

Creamos el ambiente (pilas)

Las expresiones with introducen al ambiente cada asignación de variable con su valor.

Las funciones ahora se almacenan como closures.



Creamos el ambiente (listas)

$$\text{env0} = \quad (\ )$$

$$\text{env1} = \quad (\ \underbrace{(x\ \ 3)}_{\text{aSub}}\ )$$

$$\text{env2} = \quad (\ \underbrace{(\text{foo}\quad \{\text{closureV}\ \ y\ \ \{+x\ y\}\ \text{env1}\})}_{\text{aSub}}\ \ \underbrace{(x\ \ 3)}_{\text{aSub}}\ )$$

$$\text{env3} = \quad (\ \underbrace{(x\ \ 5)}_{\text{aSub}}\ \ \underbrace{(\text{foo}\quad \{\text{closureV}\ \ y\ \ \{+x\ y\}\ \text{env1}\})}_{\text{aSub}}\ \ \underbrace{(x\ \ 3)}_{\text{aSub}}\ )$$

Las expresiones with introducen al ambiente cada asignación de variable con su valor.

Las funciones ahora se almacenan como closures.

Ejemplo 2 (línea 2)

```
(interp   {foo   4}   env3 )

[app ( foo    4)

  fun-val   = (interp   foo   env3 )

  fun-val   = {closureV    y   {+ x y}  env1}


… ]
```

```
(define (interp expr  env)

        …

1.[app (fun-expr   arg-expr)

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

3.         (interp (closureV-body   fun-val  )

4.                   (aSub  (closureV-param   fun-val  )

5.                             (interp   arg-expr   env)

6.                              (closureV-env    fun-val  ))))]

))
```

Ejemplo 2 (línea 3)

```
[app ( foo    4)

  fun-val   = {closureV   y   {+ x  y}  env1}

(interp (closureV-body  fun-val )

             ⇒ {+ x  y}

sustituyendo:

(interp  {+  x   y}

… ]
```

```
(define (interp expr  env)

        …

1.[app (fun-expr   arg-expr)

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

3.        (interp (closureV-body   fun-val )

4.                (aSub  (closureV-param   fun-val )

5.                          (interp   arg-expr   env)

6.                           (closureV-env    fun-val ))))]

))
```

Ejemplo 2 (línea 4, 5 y 6)

```
[app ( foo    4)

 fun-val  = {closureV   y   {+ x  y}  env1}

(interp  {+  x   y}

         (aSub (closureV-param fun-val )

                  (interp   4   env3)

                 (closureV-env  fun-val]
```

```
(define (interp expr  env)

       …

1.[app (fun-expr   arg-expr)

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

3.          (interp (closureV-body   fun-val )

4.                      (aSub  (closureV-param    fun-val )

5.                                  (interp   arg-expr   env)

6.                                   (closureV-env    fun-val ))))]

))
```

Ejemplo 2 (línea 4)

```
[app ( foo   4)

 fun-val  = {closureV  y  {+ x  y}  env1}

(interp  {+  x   y}

        (aSub (closureV-param fun-val )

               ⇒   y
```

```
(define (interp expr  env)

       ...

1.[app (fun-expr   arg-expr)

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

3.         (interp (closureV-body   fun-val )

4.                   (aSub  (closureV-param   fun-val )

5.                               (interp   arg-expr   env)

6.                                (closureV-env    fun-val ))))]

))
```

Ejemplo 2 (línea 4 evaluada)

```
[app ( foo    4)

 fun-val    = {closureV   y   {+ x  y}  env1}

(interp  {+  x   y}

        (aSub   y

                (interp   4   env3)

               (closureV-env  fun-val]
```

```
(define (interp expr  env)

       ...

1.[app (fun-expr   arg-expr)

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

3.          (interp (closureV-body   fun-val )

4.                    (aSub  (closureV-param   fun-val )

5.                              (interp   arg-expr   env)

6.                               (closureV-env    fun-val ))))]

))
```

Ejemplo 2 (línea 5)

```
[app ( foo   4)

 fun-val   = {closureV   y   {+ x  y}  env1}

(interp  {+  x   y}

        (aSub   y

                (interp   4   env3)

                ⇒   4

... ]
```

```
(define (interp expr  env)

       ...

1.[app (fun-expr   arg-expr)

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

3.        (interp (closureV-body   fun-val  )

4.                  (aSub  (closureV-param   fun-val  )

5.                            (interp   arg-expr   env)

6.                             (closureV-env    fun-val  ))))]

))
```

Ejemplo 2 (línea 5 evaluada)

```
[app ( foo    4)

 fun-val   = {closureV   y   {+ x  y}  env1}

(interp  {+  x   y}

        (aSub   y    4

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

```
(define (interp expr  env)

       …

1.[app (fun-expr   arg-expr)

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

3.          (interp (closureV-body   fun-val )

4.                   (aSub  (closureV-param   fun-val )

5.                             (interp   arg-expr   env)

6.                              (closureV-env    fun-val ))))]

))
```

Ejemplo 2 (línea 6)

```
[app ( foo    4)

 fun-val   = {closureV  y   {+ x  y}  env1}

(interp  {+  x   y}

        (aSub   y   4

                (closureV-env  fun-val )

                ⇒  env1

...]
```

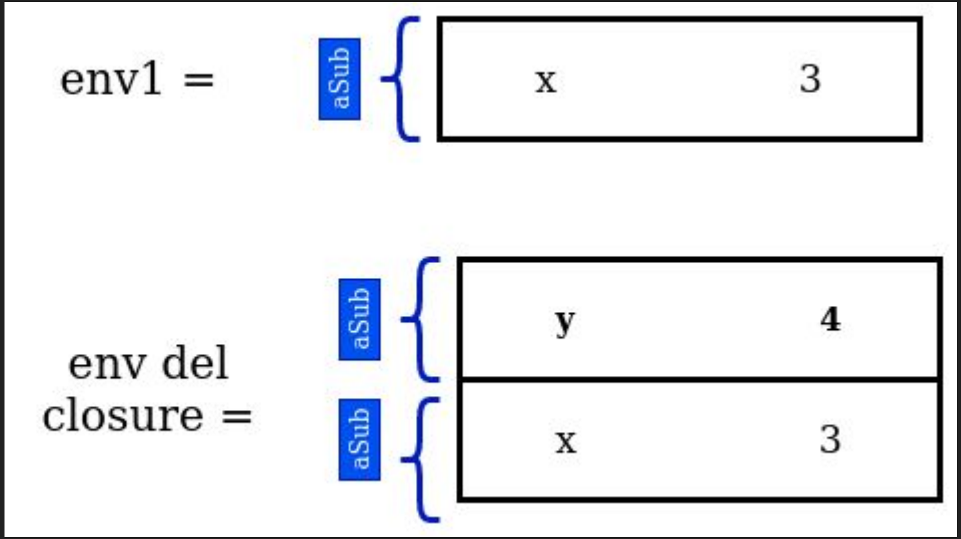
```
(define (interp expr  env)

      ...

1.[app (fun-expr   arg-expr)

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

3.        (interp (closureV-body   fun-val  )

4.                  (aSub  (closureV-param   fun-val  )

5.                            (interp   arg-expr   env)

6.                             (closureV-env    fun-val  ))))]

))
```

Ejemplo 2 (línea 6)

```
[app ( foo   4)

 fun-val   = {closureV   y   {+ x  y}  env1}

(interp  {+  x   y}

        (aSub   y   4   env1 )) ]
```

```
(define (interp expr  env)

      …

1.[app (fun-expr   arg-expr)

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

3.        (interp (closureV-body   fun-val )

4.                  (aSub  (closureV-param   fun-val )

5.                            (interp   arg-expr   env)

6.                             (closureV-env    fun-val ))))]

))
```

```
(interp {+ x  y}
        (aSub  y  4  env1 ))
```



```
[add ( x  y)
   (add-numbers (interp  x  env-del-closure)  (interp  y  env-del-closure))]
   (add-numbers 3  4 ) ⇒ (+ 3  4)  =  7    i.e.  (num  7)
```

Alcance estático

```
{with {x  3}

    {with {foo  {fun {y} {+  x  y}}}

        {with {x   5}

            {foo   4}}}}
```

```
{ foo   4 }

{ {fun {y} {+  x  y}}    4}

{fun {4} {+  3  4}}

= {+  3  4}  =  7
```

GRACIAS