Lenguajes de programación

Oscar Eduardo Galaviz Cuen

7 de Septiembre de 2022

- Exercise 3.20 [⋆] In PROC, procedures have only one argument, but one can get the effect of multiple argument procedures by using procedures that return other procedures. For example, one might write code like

```
let f = proc (x) proc (y) ...
in ((f 3) 4)
```

  This trick is called Currying, and the procedure is said to be Curried. Write a Curried procedure that takes two arguments and returns their sum. You can write x+y in our language by writing $-(x, -(0, y))$.

```
let f = proc (x) proc (y) -(x, -(0,y))
in ((f 3) 4)
```

- Exercise 3.27 [⋆] Add a new kind of procedure called a traceproc to the language. A traceproc works exactly like a proc, except that it prints a trace message on entry and on exit.
  Sintaxis concreta:
  Expressión ::= **traceproc** (*Identifier*) *Expression*
  Sintaxis abstracta:
  (traceproc-exp var body)
  En la semantica se mantienen los mismos valores expresados y denotados.

```
(value-of (traceproc-exp var body) env)
= (proc-val (procedure var body env))
```

  Cambios en procedure:

```
(define-datatype proc proc?
    (procedure
        (var identifier?)
        (body expression?)
        (saved-env environment?))
    (trace-procedure
        (var identifier?)
        (body expression?)
        (saved-env environment?)))
```

Cambios en apply-procedure:

```
(define (apply-procedure proc1 val)
    (cases proc proc 1
        (procedure (var body env)
            (valueof-body [var = val]env))
        (trace-procedure (var body env)
            (display "Entering proc...")
            (valueof-body [var = val]env)
            (display "Exiting proc..."))))
```

- Exercise 3.28 [⋆⋆] Dynamic binding(or dynamic scoping) is an alternative design for procedures, in which the procedure body is evaluated in an environment obtained by extending the environment at the point of call. For example in

```
let a = 3
in let p = proc (x) -(x,a)
       a = 5
    in -(a, (p 2))
```

the a in the procedure body would be bound to 5, not 3. Modify the language to use dynamic binding. Do this twice, once using a procedural representation for procedures, and once using a data-structure representation.
Cómo esta vez el ambiente es extendido hasta la llamada a función, entonces proc-val ya no va a recibir un ambiente, por ende:

```
(value-of (proc-exp var body) env)
    = (proc-val (procedure var body))
```

Con esto, ya nos damos cuenta que cambia el value-of de call-exp, pues este, al obtener el valor de tipo procedimiento, de la expresión evaluada de op-exp, este ya no recibe un ambiente, y a falta de este, ahora se tiene que enviar en apply-procedure.

```
  (value-of (call-exp op-exp arg-exp) env)
    =(let ([proc (expval->proc (value-of op-exp env))]
           [arg (value-of arg-exp env)])
        (apply-procedure proc arg env))
donde:
  (apply-procedure (procedure var body) val env)
     =(value-of body [var = val]env)
```

  - Representación con procedimientos:
    Cambios en procedure:

```
(define (procedure var body)
    (value-of body env)
```

Cambios en apply-procedure:

```
(define (apply-procedure proc1 val env)
(proc1 val [var = val]env))
```

– Representación con estructuras de datos:
Cambios en procedure:

```
(define-datatype proc proc?
    (procedure
        (var identifier?)
        (body expression?)))
```

Cambios en apply-procedure:

```
(define apply-procedure proc val env
    (cases proc proc1
        (procedure (var body)
            (value-of body [var = val]env))))
```