Ejercicios PROC
Miguel Sebastian Navarro Islas

Ejercicio 3.20 En PROC, los procedimientos solo tienen un argumento, pero se puede obtener el efecto de argumentos multiples usando procedimientos que regresan otros procedimientos. A este truco se le llama *Currying*. Escribe un procedimiento currificado que tome dos argumentos y regrese su suma.

```
let f = proc (x) proc (y) -(x -(0 y))
```

Ejercicio 3.26 En la representacion de los procedimientos se guarda el entorno completo en la clausura. Sin embargo, solo necesitamos los bindings para las variables libres. Modifica la representacion de los procedimientos para guardar solo las variables libres.
Modificaciones en la implementacion:

```
(proc-val (procedure var body (free-vars env var body)))

(define (free-vars env var body)
   (let loop ([free-vars '()]
                  [var var]
                  [body body])
          (cond
            [(const-exp? exp)
             free-vars]
            [(var-exp? exp)
             (let ([var2 (var-exp-var exp)])
               (if (eq? var var2)
                   free-vars
                   (extend-env var2 (apply-env env var2) free-vars)))]
            [(diff-exp? exp)
             (let ([exp1 (diff-exp-exp1 exp)]
                   [exp2 (diff-exp-exp2 exp)])
               (loop (loop free-vars var exp1) var exp2))]
            [(zero?-exp? exp)
             (let ([exp1 (zero?-exp-exp1 exp)])
               (loop free-vars var exp1))]
            [(if-exp? exp)
             (let ([exp1 (if-exp-exp1 exp)]
                   [exp2 (if-exp-exp2 exp)]
                   [exp3 (if-exp-exp3 exp)])
               (loop (loop (loop free-vars var exp1) var exp2) var exp3))]
            [(let-exp? exp)
             (let ([var (let-exp-var exp)]
                   [exp1 (let-exp-exp1 exp)]
                   [body (let-exp-body exp)])
               (loop (loop free-vars var exp1) var body))]
            [(proc-exp? exp)
             (let ([var (proc-exp-var exp)]
                   [body (proc-exp-body exp)])
               (loop free-vars var body))]
            [(call-exp? exp)
```

```
            (let ([rator (call-exp-rator exp)]
                  [rand (call-exp-rand exp)])
              (loop (loop free-vars var rator) free-vars var rand))])))
```

Ejercicio 3.27 Agrega un nuevo tipo de procedimiento llamado traceproc. Este funciona exactamente igual que proc pero imprime un mensaje de rastreo en la entrada y salida del procedimiento.
Modificacion a la sintaxis concreta:

```
Expression := trace-proc (Identifier) Expression
```

Modificacion a la sintaxis abstracta:

```
(traceproc-exp var body)
```

Semantica:
Se mantienen los mismos valores expresados y denotados, modificacion a la semantica:

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

Cambios en la funcion *apply-procedure*:

```
(define (apply-procedure proc val)
  (unless (procedure? proc)
    (error 'value-of "no es un procedimiento: ~e" proc))
  (let ([var (procedure-var proc)]
        [body (procedure-body proc)]
        [saved-env (procedure-saved-env proc)])
  (cond
   [(traceproc-exp? proc)
      (display "Entering proc...")
      (value-of body (extend-env var val saved-env))
      (display "Exiting proc...")]
        [(proc-exp? proc)
         (value-of body (extend-env var val saved-end))])))
```