Ejercicios PROC

Enrique Giottonini

Octubre 12, 2022

Exercise 3.19 [⋆]

In many languages, procedures must be created and named at the same time. Modify the language of this section to have this property by replacing the `proc` expression with a `letproc` expression.

Sintáxis Concreta y Abstracta

Expression ::= ~~proc (Identifier) Expression~~
Expression ::= letproc Identifier (Identifier) Expression in Expression

(letproc-exp name param body exp1)

Semántica

```
(value-of (letproc-exp name param body exp1) env) =
(value-of exp1 ([name = (procedure param body env)] env))
```

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

Exercise 3.21 [⋆⋆]

Extend the language of this section to include procedures with multiple arguments and calls with multiple operands, as suggested by the grammar

$$\text{Expression} ::= \text{proc } (\{\text{Identifier}\}^{*(,)}) \text{ Expression}$$
$$\text{Expression} ::= (\text{Expression } \{\text{Expression}\}^{*(,)})$$

Sintáxis Abstracta

(proc-exp params body)
(call-exp fun args)

Semántica

```
(value-of (proc-exp params body) env)
=
(cond
   [(empty? params) (proc-val (procedure unreach body env))]
   [(= (length? params) 1) (proc-val (procedure (car params) body env))]
   [else (proc-val (procedure
                       (car params)
                       (proc-exp (cdr params) body) env)
                       env)])


(value-of (call-exp fun args) env)
=
(let ((proc (expval->proc (value-of fun env))))
  (cond
     [(empty? args) (value-of (procedure-body proc) (procedure-env proc))]
     [(= (length? args) 1) (apply proc (value-of (car args) env))]
     [else
        (value-of (call-exp (call-exp (car params)) (cdr params)))]))
```

Exercise 3.23 [⋆⋆]

What is the value of the following PROC program?

```
⌈ <<let makemult = proc (maker)
                     proc (x)
                       if zero?(x)
                       then 0
                       else -(((maker maker) -(x,1)), -4)
        in let times4 = proc (x) ((makemult makemult) x)
           in (times4 3)>> empty-env⌉
=
⌈ << let times4 = proc (x) ((makemult makemult) x)
     in (times4 3)>>
     [makemult = (procedure
                     maker
                     proc (x) if zero?(x)
                                 then 0
                                 else -(((maker maker) -(x,1)), -4)]
                     empty-env]⌉
=
⌈ << let times4 = proc (x) ((makemult makemult) x)
     in (times4 3)>>
     makemult-env ⌉
=
⌈ <<(times4 3)>>[times4 = (procedure
                              x
                              ((makemult makemult) x)
                              makemult-env] makemult-env⌉
## saltando directamente al return del apply-procedure ##
=
⌈ <<((makemult makemult) x)>> [x=3]
                                 makemult-env⌉
=
(apply (expval->proc
           ⌈<<(makemult makemult)>>[x=3] makemult-env⌉)
       3)
=
(apply (expval->proc
           (apply (expval->proc ⌈<<makemult>>[x=3] makemult-env⌉)
                  ⌈<<makemult>>[x=3] makemult-env⌉)
       3)
=
(apply (expval->proc
           (apply (procedure
                     maker
                     proc (x) if zero?(x)
                              then 0
```

```
                                        else -(((maker maker) -(x,1)), -4)
                              empty-env)
                ...makemult))
        3)
=
(apply (expval->proc
               [<<proc (x) if zero?(x)
                             then 0
                      else -(((maker maker) -(x,1)), -4)>>
                    [maker=...makemult]empty-env])
        3)
=
(- ([ <<((maker maker))>> [x=2][maker=...makemult][x=3] makemult-env]
 -4)
=
...
=
(- 0
 (- 4
  (- 4 -4)))
=
12
```

Use the tricks of this program to write a procedure for factorial in PROC. As a hint, remember that you can use Currying (exercise 3.20) to define a two-argument procedure `times`.

Sea la función `times` : $N \to N$
y un entorno:

```
ρ = [times = (procedure
                    x
                    proc (y)
                      (((maketimes maketimes) x) y)
                    empty-env)
     maketimes = (procedure
                         maker
                         proc (x)
                           proc (y)
                             if zero? (y)
                             then 0
                             else (- x (- 0 ((maker maker) (- y 1)))))]
```

Una expresíon que usa factorial en un entorno ρ

```
let makefact = proc (maker)
                 proc (x)
                   if zero? (x)
                   then 1
                   else ((times x) ((maker maker) (- x 1)))
```

```
in let fact = proc (x) ((makefact makefact) x)
   in (fact 5)
```

Exercise 3.25 [⋆]

The tricks of the previous exercises can be generalized to show that we can define any recursive procedure in PROC. Consider the following bit of code:

```
let makerec = proc (f)
                let d = proc (x)
                          proc (z) ((f (x x)) z)
                in proc (n) ((f (d d)) n)
in let maketimes4 = proc (f)
                      proc (x)
                        if zero?(x)
                        then 0
                        else -((f -(x,1)), -4)
   in let times4 = (makerec maketimes4)
      in (times4 3)
```

Show that it returns 12.

`makerec` es el combinador Y, permite obtener la recursión de cualquier procedimiento.
`maketimes` es un procedimento *curryficado* recursivo para obtener la multiplicación a partir de la suma.
`times4` aplica el combinador Y a `maketimes` para hacerlo recursivo.

Mostrar que el valor del programa es 12 requiere que se apliquen las reglas sémanticas, que sería como resolver ecuaciones sustituyendo, de la misma forma que en el Ejercicio 3.23.