Your Paper

Kevin David Ruiz González

13 de octubre de 2022

1. 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))
```

2. 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)
```

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. El resultado del programa es 12 porque repetirá el mismo proceso x veces, que en este caso es 3, aumentando en 4 el resultado por cada iteración.

Sintáxis concreta:

```
let maketimes = proc (maker)
                  proc (x)
                    proc (y)
                      if zero?(x)
                      then 0
                      else -((((maker maker) -(x, 1)) y), -(0, y))
in let times = (maketimes maketimes)
   in let makefact = proc (maker)
                       proc (x)
                         if zero?(x)
                         then 1
                         else ((times x) ((maker maker) -(x, 1)))
      in (makefact makefact)
```

3. 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
El procedimiento makestimes4 toma un procedimiento times4 y devuelve un procedimiento times4. Entonces para esto convertimos maketimes4 a un procedimiento maker, que tomará un maker y devolverá un procedimiento times4 que será un contador.

```
let makerec = proc (f)
  let maker = proc (maker)
                let recursive-proc = (maker maker)
                in (f recursive-proc) x)
    in (maker maker)
```

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.

```
Sintáxis Concreta
Expression:== traceproc (Identifier) Expression
Sintáxis Abstracta
(traceproc-exp var body)

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