Problemas PROC

Ricardo Payan

11 de octubre de 2022

Ejercicio 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 suma = proc(x) proc(y) (- (x , -(0 , y))
       in (suma 5) 5)
```

Ejercicio 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
    Expression := (traceproc Identifier Expression)

#Sintaxis Abstracta
    (traceproc-exp var body)
```

```
#Especificacion semantica
    (value-of (traceproc-exp var body) env)
        = (trace (proc-val (procedure var body env)))
```

Ejercicio 3.29

Unfortunately, programs that use dynamic binding may be excep- tionally difficult to understand. For example, under lexical binding, consistently renaming the bound variables of a procedure can never change the behavior of a pro- gram: we can even remove all variables and replace them by their lexical addresses, as in section 3.6. But under dynamic binding, this transformation is unsafe. For example, under dynamic binding, the procedure $proc(z)a$ returns the value of the variable a in its caller's environment. Thus, the program

```
    let a = 3
    in let p = proc (z) a
        in let f = proc (x) (p 0)
            in let a = 5
                in (f 2)
```

returns 5, since a's value at the call site is 5. What if f's formal parameter were a?

Recordando que el formal parameter se refiere la variable var en $proc-exp(var, body)$.

Entonces cambiamos el parametro formal de f y el codigo quedaria asi.

```
let a = 3
    in let p = proc (z) a
        in let f = proc (a) (p 0)
            in let a = 5
```

```
5                    in (f 2) //Aqui es la ultima vez que se ↩
                         le asigna valor a "a"
```

Ahora se le esta asignando el valor de 2 a la variable a con la llamada del procedimiento f. Por lo tanto el valor que retornara este bloque de codigo es 2.