

# Ejercicios Proc

Francisco Javier Atondo Nubes

6/10/2022

## 1 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.

Antes -	Despues
Let f = proc (x) - (x,4) -	letproc f(x) - (x,4)
In (f (f 5)) -	in (f (f 5))

Sintaxis Concreta  
Quitamos produccion:  
Expression := proc (Identifier) Expression

Agregamos:  
Expression := leftproc Identifier (Identifier) Expression in Expression  
- Letproc-exp (fun param body exp1)

Semantica  
ExpVal = Int + Bool  
DenVal = Int + Bool + Proc  
(value-of (letproc-exp name param body exp1 ) env)  
= (value-of exp1 ( [name = (proc-val (procedure param body env) ] ) env )

## 2 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  $(x, (0, y))$ .

Sintaxis Concreta:

Definimos como:

Expression ::= proc - ( Expression x proc -( Expression y ) )

Semantica:

ExpVal = Int + Proc

(value-of exp1( proc-val ( param body ) env ) )

### 3 3.21

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

Expression ::= proc (Identifier(,)) Expression

Expression ::= (Expression Expression)

Sintaxis Concreta:

Expression ::= proc ( ( Identifier (Identifier)' ) ( Expression ( Expression ) )' )

Agregamos:

- let n

- in ( ( body )'(n) ( var )'(n) )

Sintaxis Semantica:

( call-exp ( value-of (const-exp n ) (let-exp n exp body ) ) p )

## 4 3.25

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.

## 5 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.

```
let traceproc = proc print ( procedure p )  
in let
```