**Universidad Nacional Autónoma de México**
**Facultad de Ciencias**
**Lenguajes de Programación | 7098**
Examen Parcial 04 : | Memoria y tipos
Sosa Romo Juan Mario | 320051926
22/11/24

1. **Ejercicio 1.** (25 pts.) Evalúa las siguientes expresiones usando las téctnicas de paso de prámetros que se indican. En cada caso debes mostrar cómo queda el ambiente y memoria final, tal y como se vio en clase.

   - Evalúa el siguiente código bajo paso por valor y por referencia.

```
(let [(list1 (box '(1 2 3)))
      (list2 (box '(4 5 6)))
      (concat-lists (lambda (x y)
                      (begin
                          (set! x (append (unbox x) (unbox y)))
                          (set! y '(0))))))]
      (begin
          (concat-lists list1 list2)
          (list (unbox list1) (unbox list2))))
```

   - Evalúa el siguiente código bajo paso por nombre y por necesidad.

```
(let [(acc 0)
      (conditional (lambda (x)
                      (if (> x 0)
                          (begin (set! acc (+ acc 1)) acc)
                          (begin (set! acc (- acc 1)) acc))))
      (compute (lambda (y) (* y y)))]
(compute (conditional acc)))
```

2. **Ejercicio2.** (25 pts. ) Modifica la siguiente función escrita en *Haskell* para que aplique correctamente la téctnica de memoización como vimos en clase. No puedes usar instrucciones que tengan efectos secundarios. ¿Qué hace la función? Justifica además como cambia la eficiencia en tiempo y espacio con respecto a la definición original.

```
changeWays :: Int -> [Int] -> Int
changeWays 0 _ = 1
changeWays _ [] = 0
changeWays amount (coin:coins)
    | amount < 0 = 0
    | otherwise = changeWays amount coins +
      changeWays (amount - coin) (coin:coins)
```

3. **Ejercicio 3.** (25 pts.) Realiza el juicio de tipo para cada una de las siguientes expresiones, usa las reglas vistas en clase o define una nueva regla en caso de ser necesario. Observa que la primera expresión no tiene anotaciones de tipo, por lo que tendras que definir las reglas para verificar los tipos.

(a)
```
(let (g (lambda (x) (x 4)))
     (g (lambda (y) (-y 2))))
```

(b)
```
(letrec (f : number -> number
     (fun (x : number) : number
          (if0 x 1 (* n (f (- n 1))))))
     (f 5))
```

4. **Ejercicio 4.** (25 pts.) Realiza la inferencia de tipos vista en clase sobre la siguiente expresión, recuerda obtener las restricciones y usar el algoritmo de unificación para resolverlas.

```
((lambda (x) (* x 2)) (+ 2 3))
```

**Obtener las subexpresiones**

- 1. ((lambda (x) (* x 2)) (+ 2 3))
- 2. ((lambda (x) (* x 2)) (+ 2 3))
- 3. ((lambda (x) (* x 2)) (+ 2 3))
- 4. ((lambda (x) (* x 2)) (+ 2 3))
- 5. ((lambda (x) (* x 2)) (+ 2 3))
- 6. ((lambda (x) (* x 2)) (+ 2 3))
- 7. ((lambda (x) (* x 2)) (+ 2 3))
- 8. ((lambda (x) (* x 2)) (+ 2 3))
- 9. ((lambda (x) (* x 2)) (+ 2 3))

**Obtener las restricciones**

$$
\begin{aligned}
[[2]] &= [[7]] \rightarrow [[1]] \\
[[2]] &= [[3]] \rightarrow [[4]] \\
[[3]] &= [[5]] \\
[[4]] &= number \\
[[5]] &= number \\
[[6]] &= number \\
[[7]] &= number \\
[[8]] &= number \\
[[9]] &= number
\end{aligned}
$$

**Sustituir en las restricciones (algoritmo de unificación)**

| Pila de restricciones |
| --- |
| $[[2]] = [[7]] \to [[1]]$ |
| $[[2]] = [[3]] \to [[4]]$ |
| $[[3]] = [[5]]$ |
| $[[4]] = number$ |
| $[[5]] = number$ |
| $[[6]] = number$ |
| $[[7]] = number$ |
| $[[8]] = number$ |
| $[[9]] = number$ |

| Sustituciones |
| --- |
| $\emptyset$ |

| Pila de restricciones |
| --- |
| $[[7]] \to [[1]] = [[3]] \to [[4]]$ |
| $[[3]] = [[5]]$ |
| $[[4]] = number$ |
| $[[5]] = number$ |
| $[[6]] = number$ |
| $[[7]] = number$ |
| $[[8]] = number$ |
| $[[9]] = number$ |

| Sustituciones |
| --- |
| $[[2]] := [[7]] \to [[1]]$ |

| Pila de restricciones |
| --- |
| $[[3]] = [[5]]$ |
| $[[4]] = number$ |
| $[[5]] = number$ |
| $[[6]] = number$ |
| $[[7]] = number$ |
| $[[8]] = number$ |
| $[[9]] = number$ |
| $[[7]] = [[3]]$ |
| $[[1]] = [[4]]$ |

| Sustituciones |
| --- |
| $[[2]] := [[7]] \rightarrow [[1]]$ |

| Pila de restricciones |
| --- |
| $[[4]] = number$ |
| $[[5]] = number$ |
| $[[6]] = number$ |
| $[[7]] = number$ |
| $[[8]] = number$ |
| $[[9]] = number$ |
| $[[7]] = [[5]]$ |
| $[[1]] = [[4]]$ |

| Sustituciones |
| --- |
| $[[3]] := [[5]]$ |
| $[[2]] := [[7]] \rightarrow [[1]]$ |

| Pila de restricciones |
|---|
| $[[5]] = number$ |
| $[[6]] = number$ |
| $[[7]] = number$ |
| $[[8]] = number$ |
| $[[9]] = number$ |
| $[[7]] = [[5]]$ |
| $[[1]] = number$ |

| Sustituciones |
|---|
| $[[4]] := number$ |
| $[[3]] := [[5]]$ |
| $[[2]] := [[7]] \rightarrow [[1]]$ |

| Pila de restricciones |
|---|
| $[[6]] = number$ |
| $[[7]] = number$ |
| $[[8]] = number$ |
| $[[9]] = number$ |
| $[[7]] = number$ |
| $[[1]] = number$ |

| Sustituciones |
|---|
| $[[5]] := number$ |
| $[[4]] := number$ |
| $[[3]] := number$ |
| $[[2]] := [[7]] \rightarrow [[1]]$ |

| Pila de restricciones |
|---|
| $[[7]] = number$ |
| $[[8]] = number$ |
| $[[9]] = number$ |
| $[[7]] = number$ |
| $[[1]] = number$ |

| Sustituciones |
|---|
| $[[6]] := number$ |
| $[[5]] := number$ |
| $[[4]] := number$ |
| $[[3]] := number$ |
| $[[2]] := [[7]] \rightarrow [[1]]$ |

| Pila de restricciones |
|---|
| $[[8]] = number$ |
| $[[9]] = number$ |
| $[[7]] = number$ |
| $[[1]] = number$ |

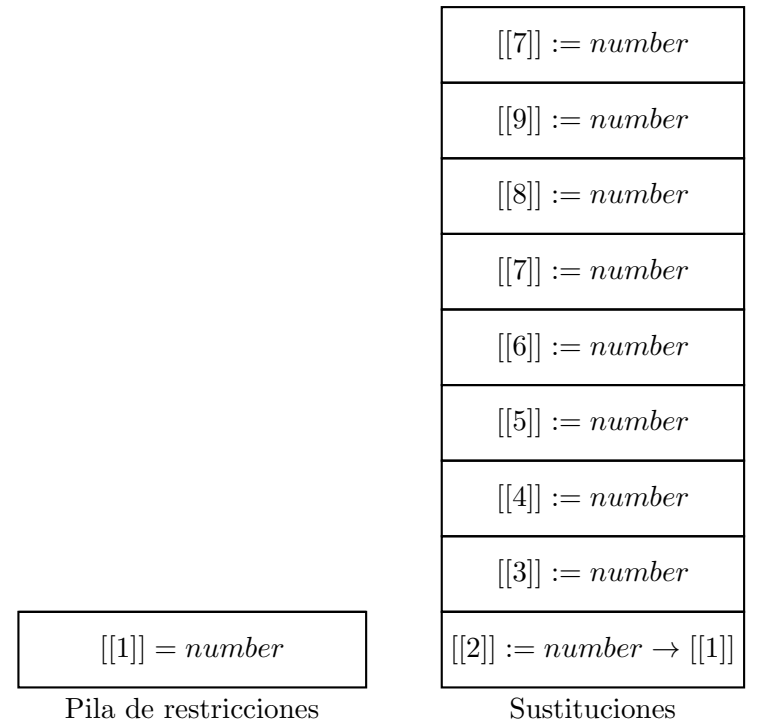
| Sustituciones |
|---|
| $[[7]] := number$ |
| $[[6]] := number$ |
| $[[5]] := number$ |
| $[[4]] := number$ |
| $[[3]] := number$ |
| $[[2]] := number \rightarrow [[1]]$ |

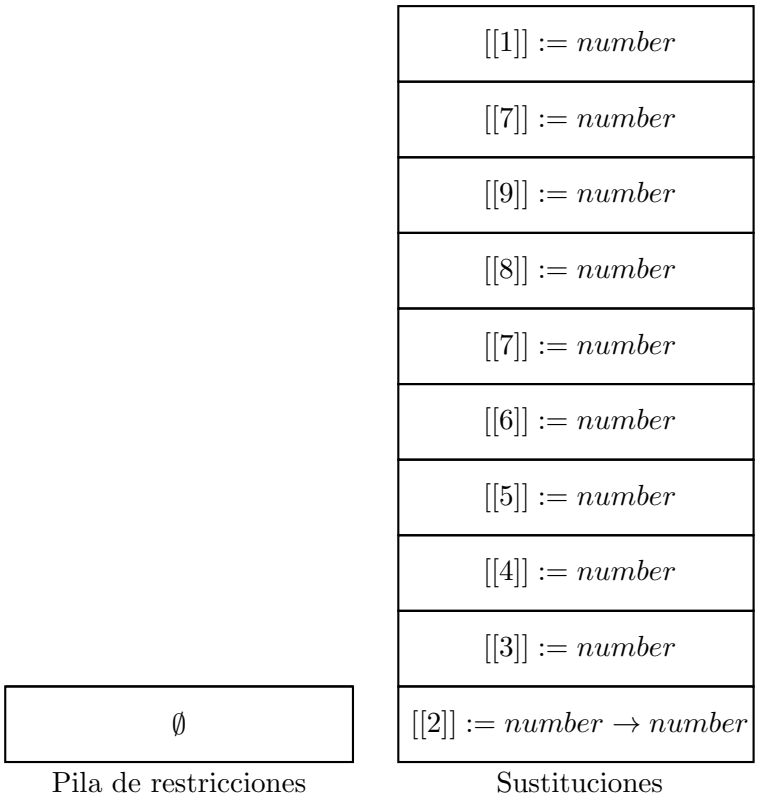
| Pila de restricciones |
|---|
| $[[9]] = number$ |
| $[[7]] = number$ |
| $[[1]] = number$ |

| Sustituciones |
|---|
| $[[8]] := number$ |
| $[[7]] := number$ |
| $[[6]] := number$ |
| $[[5]] := number$ |
| $[[4]] := number$ |
| $[[3]] := number$ |
| $[[2]] := number \rightarrow [[1]]$ |

| Pila de restricciones |
|---|
| $[[7]] = number$ |
| $[[1]] = number$ |

| Sustituciones |
|---|
| $[[9]] := number$ |
| $[[8]] := number$ |
| $[[7]] := number$ |
| $[[6]] := number$ |
| $[[5]] := number$ |
| $[[4]] := number$ |
| $[[3]] := number$ |
| $[[2]] := number \rightarrow [[1]]$ |




Que tiene sentido pues la expresión original era una aplicación que usaba la función que va de $number \to number$ y el argumento es una suma de tipo $number$ el cuerpo de la función nos regresa efectivamente un number siempre y cuando x sea un número.