

Auxiliar Tres - CC41A  
**Lenguajes de Programación**  
*Practicando con ejercicios de controles*

Oscar E. A. Callaú  
*oalvarez@dcc.uchile.cl*

Santiago - Chile, 18/Ago/2008

## 1. Representando números con funciones, C1 I-2008.

Esta pregunta tiene como objetivo que entienda como se definen los número naturales en el  $\lambda$ -cálculo<sup>1</sup>. Al respecto:

- Los número son símbolos para contar cosas. Dada la siguiente base de números

```
zero = (lambda(f)(lambda(x) x))
one  = (lambda(f)(lambda(x)(f x)))
two  = (lambda(f)(lambda(x)(f (f x))))
```

Deduzca la forma para un número  $n$  cualquiera.

**Obs:** Para que se convezca que esta representación funciona piense que  $f$  es la función `add1` y  $x$  como 0.

- Escriba la función `succ` que entrega el siguiente número al dado (en su notación funcional!)
- Escriba las funciones `add` y `mult`. Para ello note que:

$$m + n = \overbrace{1 + 1 + \dots + 1}^{n\text{-veces}} + m$$
$$m * n = \overbrace{m + m + \dots + m}^{n\text{-veces}} + 0$$

## 2. Gramáticas y s-lists, C1 I-2008

Dada la siguiente gramática concreta:

```
(define-type ALE
  [id (v VAR?)]
  [add (l ALE?) (r ALE?)]
  [sub (l ALE?) (r ALE?)])
```

---

<sup>1</sup>El  $\lambda$ -cálculo sólo cuenta con tres tipos de expresiones: variable, definición de función, y aplicación de función.

```
(define-type VAR
  [var (coef number?) (x symbol?)])
```

Implemente las funciones:

1. `parse :: s-lists -> ALE`, que parse una lista de string en un ALE.
2. `simp :: ALE -> ALE` que simplifica la expresión algebraica:

```
(simp (parse '{+ {3 a} {- {2 b} {- {1 a} {3 b}}}))
-> {+ {2 a} {5 b}}
```

Note que:

```
{3 a} = 3*a
{- a b} = a - b
```

### 3. Substitución Explícita, C1 I-2007

Considere la siguiente implementación de la función de substitución `subst` del lenguaje WAE visto en clases:

```
;; subst: WAE symbol WAE -> WAE
(define (subst expr sub-id val)
  (type-case WAE expr
    (num (n) expr)
    (add (l r) (add (subst l sub-id val)
                    (subst r sub-id val)))
    (sub (l r) (sub (subst l sub-id val)
                    (subst r sub-id val)))
    (with (bound-id named-expr bound-body)
          (if (symbol=? bound-id sub-id)
              expr
              (with bound-id
                  named-expr
                  (subst bound-body sub-id val))))
    (id (v) (if (symbol=? v sub-id) val expr))))
```

Con esta función de substitución, el calculador se cae en las dos expresiones siguientes:

1. `{with {x 5} {with {y x} y}}`
2. `{with {x 5} {with {x x} x}}`

Para cada caso explique porqué se cae el calculador, y luego de una versión corregida de `subst`.