

### Problema 1: 25%

- Escriba la reducción- $\beta$  de la operación lógica NOT
- Escriba y explique como se vería la recursión y los ciclos.
- Explique cuando es prudente usar este tipo de programación y cuando no. De un ejemplo para cada caso.

a)  $TRUE = \lambda a. \lambda b. a$

$FALSE = \lambda a. \lambda b. b$

$NOT = \lambda p. p \ FALSE \ TRUE$

→ Reducción- $\beta$  de NOT TRUE:

$NOT \ TRUE = (\lambda p. p \ FALSE \ TRUE) \ TRUE$

$\rightarrow_{\beta} \ TRUE \ FALSE \ TRUE$

$= (\lambda a. \lambda b. a) \ FALSE \ TRUE$

$\rightarrow_{\beta} (\lambda b. FALSE) \ TRUE$

$\rightarrow_{\beta} \ FALSE$

→ Reducción- $\beta$  de NOT FALSE

$NOT \ FALSE = (\lambda p. p \ FALSE \ TRUE) \ FALSE$

$\rightarrow_{\beta} \ FALSE \ FALSE \ TRUE$

$= (\lambda a. \lambda b. b) \ FALSE \ TRUE$

$\rightarrow_{\beta} (\lambda b. b) \ TRUE$

$\rightarrow_{\beta} \ TRUE$

b) En cálculo lambda puro no hay recursión directa, pero se puede lograr mediante combinadores de punto fijo, como el combinador Y:

$Y = \lambda f. (\lambda x. f(x \ x)) (\lambda x. f(x \ x))$

Con Y podemos definir recursión:

Ejemplo:

Queremos una función que cuente hasta n en Church numerals.

Definimos  $F = \lambda f. \lambda n. IF \ (n = 0) \ THEN \ 1 \ ELSE \ (f(n-1))$

Luego  $FACT = YF$

Ciclos se logran igual: un ciclo es solo recursión en la cola

c) •) Prudente:

- Cuando el problema es recursivo o de transformación de datos. Ej: compiladores, programas matemáticos
- Cuando se necesita razonamiento formal y verificación de programas. Ej: Implementar un evaluador de expresiones

•) No Prudente:

- Con rendimiento en tiempo real crítico y patrones estructurados más eficientes
  - Cuando hay mucho estado mutable y efectos secundarios
- } Ej: videojuegos de acción en tiempo real