Representación de datos con funciones

Programación funcional

Repaso

La clase pasada vimos:

- Tipos
- Funciones como valores
- Funciones de alto orden
- Currificación

Tipado de expresiones

¿Qué tipo tienen cada una de las siguientes expresiones?

- `[1,2,3]`
- `(+)`
- `($)`
- `(+) 1`
- `map`
- `map (+) [1,2,3]`
- `map ($10) (map (+) [1,2,3])`

¿Cuáles son sus formas normales?

Tipado de expresiones

¿Qué tipo tienen cada una de las siguientes expresiones?

- `[1,2,3]`
- `(+)`
- `($)`
- `(+) 1`
- `map`
- `map (+) [1,2,3]`
- `map ($10) (map (+) [1,2,3])`

¿Cuáles son sus formas normales?

Notar que `(map (+) [1,2,3])` es una lista de funciones!

Representación de datos con funciones

¿Cómo podemos representar un conjunto?

Representación de datos con funciones

¿Cómo podemos representar un conjunto?

A través de su función característica

```
type Set a = a -> Bool
```

Representación de datos con funciones

¿Cómo podemos representar un conjunto?

A través de su función característica

```
type Set a = a -> Bool
```

¿Cómo implementamos las siguientes funciones?

- `belongs :: a -> Set a -> Bool`
- `singleton :: Eq a => a -> Set a`
- `complement :: Set a -> Set a`
- `union :: Set a -> Set a -> Set a`
- `intersection :: Set a -> Set a -> Set a`
- `image :: Set (a,b) -> a -> Set b`
- `diagonal :: Eq a => Set (a,a)`

Halting-Problem

Desafío

¿Cómo definimos las siguientes funciones?

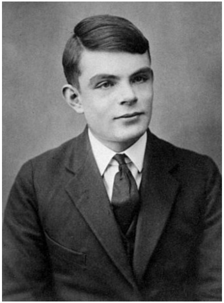
- `(==) :: Set a -> Set a -> Bool`
- `cardinal :: Set a -> Int`

Halting-Problem

Desafío

¿Cómo definimos las siguientes funciones?

- `(==) :: Set a -> Set a -> Bool`
- `cardinal :: Set a -> Int`



Alan Mathison Turing

No podemos. Esto equivale a escribir una función computable capaz de predecir si la ejecución de un programa termina.

Concepto de la prueba

Supongamos que existe una función computable `halts`:

```
halts :: (a -> b) -> Bool
```

Entonces consideremos:

```
g = if halts g then bottom else True
```

¿Qué sucede con `(halts g)`?

- Si `(halts g)` es `True`, g loopea infinitamente (`bottom`).
- Si `(halts g)` es `False`, g retorna `True`.

Absurdo, que proviene de suponer que existe `halts` computable. (La prueba formal utiliza el argumento diagonal de Cantor y excede el alcance de este curso)