Prueba de Oposición

AY2 - Teoría

Teodoro Freund

5 de Noviembre del 2018
11:00

El Ejercicio 1.19 (retocado)

Se cuenta con la siguiente representación de conjuntos `type Conj a = (a->Bool)` caracterizados por su función de pertenencia. De este modo, si `c` es un conjunto y `e` un elemento, la expresión `c e` devuelve `True` sii `e` pertenece a `c`.

El Ejercicio 1.19 (retocado)

Se cuenta con la siguiente representación de conjuntos `type Conj a = (a->Bool)` caracterizados por su función de pertenencia. De este modo, si `c` es un conjunto y `e` un elemento, la expresión `c e` devuelve `True` sii `e` pertenece a `c`.

(III) Definir un conjunto de funciones que contenga infinitos elementos, pero no todos, y dar su tipo. Además, demuestre (en Haskell) que tiene infinitas funciones pero no todas.

Contexto

Materia: Paradigmas de Lenguajes de Programación

Contexto

Materia: Paradigmas de Lenguajes de Programación

Programación Funcional, `Haskell`

Contexto

Materia: Paradigmas de Lenguajes de Programación

Programación Funcional, `Haskell`

Primera Parte de la materia:

> Los alumnos ya retomaron `Haskell`, reforzando su sintáxis y las herramientas que nos provee.
> Ya practicaron con esquemas de recursión como `fold`.

¿Por qué?

- Es un ejercicio que ejemplifica muchas cositas de Haskell que no se ven tanto en la materia. Es una buena forma de decir *"Terminamos con Haskell, pero Haskell no terminó"*.
- Da para hablar un montón de temas, entre ellos:
  + la semántica de `undefined`,
  + contravarianza,
  + typeclasses (`Eq`, `Foldable`),
  + cosas infinitas y Haskell,
  + etc.
- Es un buen ejercicio para dar en clase y analizarlo en profundidad.

El Ejercicio 1.19 (retocado), de nuevo

Se cuenta con la siguiente representación de conjuntos `type Conj a = (a->Bool)` caracterizados por su función de pertenencia. De este modo, si `c` es un conjunto y `e` un elemento, la expresión `c e` devuelve `True` sii `e` pertenece a `c`.

(III) Definir un conjunto de funciones que contenga infinitos elementos, pero no todos, y dar su tipo. Además, demuestre (en Haskell) que tiene infinitas funciones pero no todas.

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene?

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene?

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible?

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible? Definamos una función, para la cual no lo pueda decidir.

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible? Definamos una función, para la cual no lo pueda decidir.

¿Podemos hacerlo `Conj (a -> Int)` y `Conj (Int -> a)`?

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible? Definamos una función, para la cual no lo pueda decidir.

¿Podemos hacerlo `Conj (a -> Int)` y `Conj (Int -> a)`? Cómo genero algún valor de tipo `a`.

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible? Definamos una función, para la cual no lo pueda decidir.

¿Podemos hacerlo `Conj (a -> Int)` y `Conj (Int -> a)`? Cómo genero algún valor de tipo `a`. Cómo evalúo un valor de tipo `a`.

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible? Definamos una función, para la cual no lo pueda decidir.

¿Podemos hacerlo `Conj (a -> Int)` y `Conj (Int -> a)`? Cómo genero algún valor de tipo `a`. Cómo evalúo un valor de tipo `a`.

¿Y `Conj (a -> b)`?

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible? Definamos una función, para la cual no lo pueda decidir.

¿Podemos hacerlo `Conj (a -> Int)` y `Conj (Int -> a)`? Cómo genero algún valor de tipo `a`. Cómo evalúo un valor de tipo `a`.

¿Y `Conj (a -> b)`? Cómo hago todo eso a la vez.

(III) Conjunto infinito de funciones

```
infinitos :: Conj (Int -> Int)
infinitos f = f 0 `mod` 2 == 0
```

¿Cuantos elementos tiene? Creemos una lista de infinitos elementos que están.

¿Tiene a todos los elementos? ¿Cuantos no tiene? Creemos una lista de inf elementos que no están.

¿Es decidible? Definamos una función, para la cual no lo pueda decidir.

¿Podemos hacerlo `Conj (a -> Int)` y `Conj (Int -> a)`? Cómo genero algún valor de tipo `a`. Cómo evalúo un valor de tipo `a`.

¿Y `Conj (a -> b)`? Cómo hago todo eso a la vez.

¿Puedo agregarle elementos?