Taller de Álgebra I

Clase 3 - Recursión

Primer cuatrimestre 2022

Recursión

- Hasta ahora, especificamos funciones que consistían en "expresiones sencillas".

Recursión

- Hasta ahora, especificamos funciones que consistían en "expresiones sencillas".
- ¿Cómo es una función en Haskell para calcular el factorial de un número $n \in \mathbb{N}_0$?

Recursión

- Hasta ahora, especificamos funciones que consistían en "expresiones sencillas".
- ¿Cómo es una función en Haskell para calcular el factorial de un número $n \in \mathbb{N}_0$?

$$n! = \prod_{k=1}^{n} k \qquad\qquad n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

Recursión

- Hasta ahora, especificamos funciones que consistían en "expresiones sencillas".
- ¿Cómo es una función en Haskell para calcular el factorial de un número $n \in \mathbb{N}_0$?

$$n! = \prod_{k=1}^{n} k \qquad\qquad n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

¡La segunda definición de `factorial` involucra a esta misma función del lado derecho!

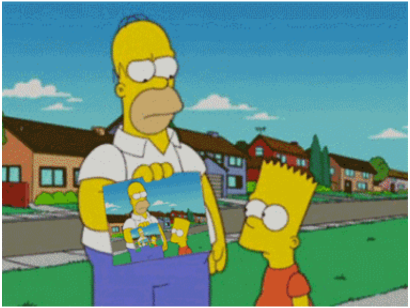
Recursión

- Hasta ahora, especificamos funciones que consistían en "expresiones sencillas".
- ¿Cómo es una función en Haskell para calcular el factorial de un número $n \in \mathbb{N}_0$?

$$n! = \prod_{k=1}^{n} k \qquad\qquad n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

¡La segunda definición de `factorial` involucra a esta misma función del lado derecho!

```
factorial :: Int -> Int
factorial n
      | n == 0 = 1
      | n > 0 = n * factorial (n-1)
```



Recursión y reducción

¿y si estaba definida así?:

```
factorial :: Int -> Int
factorial n
  | n == 0 = 1
  | otherwise = n * factorial (n-1)
```

Pattern matching:

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n-1)
```

Asegurarse de llegar a un caso base

- Consideremos este programa recursivo para determinar si un entero positivo es par:

- ```
  esPar :: Int -> Bool
  esPar n | n==0 = True
          | otherwise = esPar (n-2)
  ```

  ¿Qué problema tiene esta función?
  ```

Asegurarse de llegar a un caso base

- Consideremos este programa recursivo para determinar si un entero positivo es par:

- ```
  esPar :: Int -> Bool
  esPar n | n==0 = True
          | otherwise = esPar (n-2)
  ```

  ¿Qué problema tiene esta función?

  ¿Cómo se arregla?
  ```

Asegurarse de llegar a un caso base

- Consideremos este programa recursivo para determinar si un entero positivo es par:
- ```
  esPar :: Int -> Bool
  esPar n | n==0 = True
          | otherwise = esPar (n-2)
  ```

  ¿Qué problema tiene esta función?

  ¿Cómo se arregla?
- ```
  esPar :: Int -> Bool
  esPar n | n==0 = True
          | n==1 = False
          | otherwise = esPar (n-2)
  ```
- ```
  esPar :: Int -> Bool
  esPar n | n==0 = True
          | otherwise = not (esPar (n-1))
  ```
  ```

Definiciones recursivas

¿Cómo pensar recursivamente?

- Si queremos definir una función recursiva, por ejemplo `factorial`,
  - en el paso recursivo, **suponiendo** que tenemos el resultado para el caso anterior, ¿qué falta para poder obtener el resultado que quiero?
    En este caso, suponemos ya calculado `factorial (n-1)` y lo combinamos multiplicándolo por `n` para lograr obtener `factorial n`.

Definiciones recursivas

¿Cómo pensar recursivamente?

- Si queremos definir una función recursiva, por ejemplo `factorial`,
  - en el paso recursivo, **suponiendo** que tenemos el resultado para el caso anterior, ¿qué falta para poder obtener el resultado que quiero?
    En este caso, suponemos ya calculado `factorial (n-1)` y lo combinamos multiplicándolo por `n` para lograr obtener `factorial n`.
  - además, identificamos el o los casos base. En el ejemplo de `factorial`, definimos como casos base la función sobre 0: `factorial n | n == 0 = 1`

Definiciones recursivas

¿Cómo pensar recursivamente?

- Si queremos definir una función recursiva, por ejemplo `factorial`,
  - en el paso recursivo, **suponiendo** que tenemos el resultado para el caso anterior, ¿qué falta para poder obtener el resultado que quiero?
    En este caso, suponemos ya calculado `factorial (n-1)` y lo combinamos multiplicándolo por `n` para lograr obtener `factorial n`.
  - además, identificamos el o los casos base. En el ejemplo de `factorial`, definimos como casos base la función sobre 0: `factorial n | n == 0 = 1`
- Propiedades de una definición recursiva:
  - las llamadas recursivas tienen que *"acercarse"* a un caso base.
  - tiene que tener uno o más casos base que dependerán del tipo de llamado recursivo.
    Un caso base, es aquella expresión que no tiene paso recursivo.

Más funciones recursivas

Ejercicios

1. Implementar la función $fib : \mathbb{Z}_{\geq 0} \to \mathbb{Z}$ que devuelve el i-ésimo número de Fibonacci. Recordar que la secuencia de Fibonacci se define como:

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{en otro caso} \end{cases}$$

2. Implementar una función `parteEntera :: Float -> Integer` que calcule la parte entera de un número real positivo.

Más funciones recursivas

Ejercicios

1. Escribir una función para determinar si un número natural es múltiplo de 3. No está permitido utilizar `mod` ni `div`.
2. Implementar la función `sumaImpares :: Int -> Int` que dado $n \in \mathbb{N}$ sume los primeros n números impares. Ej: `sumaImpares 3` $\rightsquigarrow$ `1+3+5` $\rightsquigarrow$ 9.
3. Escribir una función `medioFact` que dado $n \in \mathbb{N}$ calcula $n!! = n\,(n-2)(n-4)\cdots$. Por ejemplo:

   `medioFact` 10 $\rightsquigarrow 10 * 8 * 6 * 4 * 2 \rightsquigarrow 3840$.
   `medioFact` 9 $\rightsquigarrow 9 * 7 * 5 * 3 * 1 \rightsquigarrow 945$.

Y más funciones recursivas

Ejercicios

1. Escribir una función que determine la suma de dígitos de un número positivo. Para esta función pueden utilizar `div` y `mod`.
2. Implementar una función que determine si todos los dígitos de un número son iguales.