Valores, expresiones y reducción

Programación funcional

Introducción

Docentes:

- Daniel Ciolek
- Federico Sawady

Cursada:

- Teóricas: Miércoles de 15 a 17
- Prácticas: Sábados de 11 a 13

Sitio:

- `https://sites.google.com/site/pftpiunq/`

Evaluación:

- Parcial / Recuperatorio
- TP
- Promoción (7+) / Integrador

Preguntas

¿Qué es un programa?

¿Qué es un modelo de cómputo?

¿Cuáles son las propiedades deseables de un programa?

Preguntas

¿Qué es un programa?

Programa

Descripción ejecutable.

¿Qué es un modelo de cómputo?

¿Cuáles son las propiedades deseables de un programa?

Preguntas

¿Qué es un programa?

Programa

Descripción ejecutable.

¿Qué es un modelo de cómputo?

Modelo de cómputo

Modelo formal que explica el proceso por el cuál a partir de un input se obtiene un output.

¿Cuáles son las propiedades deseables de un programa?

Preguntas

¿Qué es un programa?

Programa

Descripción ejecutable.

¿Qué es un modelo de cómputo?

Modelo de cómputo

Modelo formal que explica el proceso por el cuál a partir de un input se obtiene un output.

¿Cuáles son las propiedades deseables de un programa?

- Correctitud
- Eficiencia
- Claridad/Simplicidad
- Modificabilidad/Extensibilidad
- Otras...

Programación funcional

Alonso Church

Paradigma de programación inspirado en el lambda cálculo.

Repaso:

Definición de elementos:

```
data Day  = Mon | Tue | Wed | Thu | Fri | Sat | Sun
data Bool = False | True
data Nat  = Zero | Suc Nat
```

Definición de funciones:

```
isWeekend Sat = True
isWeekend Sun = True
isWeekend _   = False

not True  = False
not False = True

inc n = Suc n
```

Visión denotacional:

- **Valores:** elementos abstractos.
- **Expresiones:** construcciones sintácticas (correctas) que denotan valores.

Sintaxis:

Términos (`T`):

- Variables: `x`
- Funciones: `(\x -> T)`
- Aplicación: `T1 T2`
- Pattern-Matching[1]: `case T of {C1 -> T1; ...; CN -> TN}`

Syntactic sugar:

```
f x = T  ≡  f = \x -> T

g C1 = T1
g C2 = T2  ≡  g = \x -> case x of {C1 -> T1; C2 -> T2}

if T1 then T2 else T3  ≡  case T1 of {True -> T2; False -> T3}
```

[1] Versión simplificada.

Variables libres vs ligadas

Una variable se dice "ligada" si aparece dentro del alcance de un lambda, en caso contrario contrario se dice "libre".

Ejemplo:

```
(\x -> x) y  --  x esta ligada, y esta libre
```

! Una expresión con variables libres es inválida.

Reducción: reglas de reescritura

$$\frac{D \cup \{\texttt{x = T}\}, \texttt{x}}{\texttt{T}} \text{ Expansión de defición}$$

$$\frac{D, \texttt{(\x -> T1) T2}}{\texttt{T1[x <- T2]}} \text{ Beta reducción}^2$$

$$\frac{D, \texttt{case Ci of \{C1 -> T1; ...; Cn -> Tn\}}}{\texttt{Ti}} \text{ Matching}^2$$

Forma Normal (FN)

Término que no se puede reducir más.

[2] Versión simplificada

Reducción: pattern-matching

Un pattern puede contener parámetros que se substituyen en el término resultado.

$$\frac{D, \texttt{case (Ci T) of \{C1 x -> T1; ...; Cn x -> Tn\}}}{\texttt{Ti[x <- T]}}\ \text{Matching 1}$$

$$\frac{D, \texttt{case (Ci Ta Tb) of \{C1 x y -> T1; ...; Cn x y -> Tn\}}}{\texttt{Ti[x <- Ta, y <- Tb]}}\ \text{Matching 2}$$

! Como syntatic sugar Haskell admite patterns anidados.
! Cada parámetro debe ser una variable fresca (distinta).

Reducción: selección del redex

- Eager o Aplicativo: Primero lo más interno (eg. argumentos).
- Lazy o Normal: Primero lo más externo.
- Otros...

```
zero x = Zero

inf = Suc inf

zero inf  --  a que reduce?
```

Reducción: propiedades

- **Normalización:** Si se llega a una FN, es siempre la misma independientemente del orden de reducción.
- **Confluencia:** El orden normal llega a una FN si existe al menos una manera de llegar a una FN.
- **Transparencia referencial:** El resultado de las funciones depende sólo de sus parámetros.

Desafío

¿Cómo podemos definir una función aleatoria `random`?