Lenguajes de Programación

Introducción al Lenguaje Funcional Haskell

Área Fundamentos Teóricos
Departamento de Teoría de la Computación
Facultad de Informática
Universidad Nacional del Comahue

Introducción

- El diseño de los *lenguajes imperativos* está basado directamente en la *arquitectura von Neumann*
  - Eficiencia: consideración principal
  - No se considera la adecuación del lenguaje de programación al desarrollo de software
- El diseño de los *lenguajes funcionales* está basado en las *funciones matemáticas*
  - Con una sólida base teórica cercana al usuario
  - Alejada de la arquitectura de las computadoras en las cuales el programa va a ejecutarse

Antecedentes

- Existen distintos "estilos" para programar
  - Programación funcional
  - Programación lógica
  - Programación orientada a objetos
- Se desarrollaron distintos lenguajes para "adherir" a los diferentes "estilos"
  - Cada uno se basa en un modelo distinto de computación, diferente al modelo de von Neumann

Antecedentes

- Programación Funcional:
  - Provee una visión uniforme de los programa como "funciones"
  - Trata las funciones como datos
  - Previene los efectos colaterales
- Lenguajes Funcionales
  - Semántica más simple
  - Modelo de computación más sencillo
  - Usado en prototipado rápido, inteligencia artificial, sistemas de prueba matemática, aplicaciones lógicas

Antecedentes

- Hasta hace poco, la mayoría de los lenguajes funcionales eran poco eficientes (interpretados en vez de compilados)
- Hoy son muy atractivos:
  - Permiten paralelismo con simplicidad
  - Son más eficientes que los imperativos en contextos multi-procesadores.
  - Tienen librerías de aplicación ya maduras

Pero...

- Los LPF no son muy usados:
  - Los programadores no los aprenden o aprenden inicialmente programación imperativa y/o orientada a objetos
  - LPOO tienen principios organizantes fuertes para estructurar el código que imita los objetos del mundo real
- Características funcionales se han implementado en muchos lenguajes:
  - Recursión, abstracción funcional, funciones de alto nivel, etc.

“Programas” como “Funciones”

- Programa: descripción de una computación específica
- Ignorar el “como” y focalizarse en el resultado, en el “que” de la computación
  - Programa = caja negra
  - Transforma entrada en salida
  - Programa = función matemática

“Programas” como “funciones”

- **Definición Funcional:** describe como un valor se calcula, a partir de sus parámetros
- **Aplicación Funcional**: una llamada a una función definida, usando los parámetros reales (o los valores de los parámetros formales para una computación determinada)
- En matemática, no hay diferencia esencial entre parámetros y variables

Haskell

- http://www.haskell.org

¿Porqué Haskell?

- Haskell es un lenguaje de "***muy alto nivel***" (muchos de sus detalles se resuelven automáticamente)
- Haskell es ***expresivo*** y ***consiso*** (puede hacer mucho con poco esfuerzo).
- Haskell es excelente manipulando datos ***complejos*** y combinando sus componentes
- Haskell ***no*** es un lenguaje de alta performance: *prioriza el tiempo del programador y no el tiempo de la computadora/ejecución*

¿Porqué Haskell?

- Adecuado tanto para la enseñanza, investigación y para las aplicaciones
- Tiene una descripción completa con una sintaxis y semántica formal.
- Disponibilidad (es libre/gratuito)
- Basa sus ideas en un amplio consenso
- Funcional puro

Haskell: herramientas

- Hugs:
  - Intérprete rápido para cargar y ejecutar
  - Simple de usar
  - Presente en casi todas las plataformas
  - Casi sin mantenimiento
- GHC (Glasgow Haskell Compiler)
  - Compilador eficiente y seguro
  - Simple de Usar
  - Casi "estándar de facto"
  - Ejecución muy rápida
  - Requiere más recursos que Hugs.

Haskell: herramientas

- Otros
  - Nhc98
  - HBC / HBI
- Consultar
  - www.haskell.org/implementations.html

Haskell

- Ejemplo Interactivo
  - Como calculadora
- Ejemplo simple
  - Haskell1.hs
    - add
    - add10
    - Ejemplos con signaturas
    - Ejemplos de seguridad en tipos de datos
  - Definir función infija

ejemplo1.hs

```
sumar x y = x + y + 100

minumero = 15
```

```
/Ejemplos haskell$ ghci
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> :l ejemplo1.hs
[1 of 1] Compiling Main             ( ejemplo1.hs, interpreted )
Ok, modules loaded: Main.
*Main> sumar minumero minumero
130
*Main>
```

haskell1.hs

```haskell
add_1 x y = x + y

add10 x y = x + y + 10

sumar :: Int -> Int -> Int
sumar x y
   | x > y = x + y
   | otherwise = x + y + 100
--   | x < y
--   | x == y =
addf :: Float -> Float -> Float
addf x y
   | x > y = x + y
   | otherwise = x + y + 100
```

haskell1.hs

```
*Main> :l haskell1.hs
[1 of 1] Compiling Main             ( haskell1.hs, interpreted )
Ok, modules loaded: Main.
*Main> sumar 10 20
130
*Main> sumar 20 10
30
*Main> sumar 20.0 10

<interactive>:7:7:
    No instance for (Fractional Int) arising from the literal `20.0'
    Possible fix: add an instance declaration for (Fractional Int)
    In the first argument of `sumar', namely `20.0'
    In the expression: sumar 20.0 10
    In an equation for `it': it = sumar 20.0 10
*Main>
```

Haskell

- Definición de Funciones
  - Por patrones
  - esCero
  - fib
- Mostrar ejecuciones en pantalla

haskell2.hs

```haskell
(&&&) :: Int -> Int -> Int
x &&& y
    | x > y = y
    | otherwise = x

-- Definición por patrón

esCero :: Int -> Bool
esCero 0 = True
esCero _ = False

fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n
  | n > 1 = fib (n-2) + fib (n-1)
  | otherwise = 0
```

haskell3.hs

```haskell
-- Trae una función de la librería de Caracteres

import Data.Char (ord)
import Data.Char (chr)

offset = ord 'A' - ord 'a'

mayusculas :: Char -> Char
mayusculas ch = chr ( ord ch + offset)


sumador x y = x + 1
```

haskell3.hs

- Haskell es perezoso
- No necesita evaluar el segundo argumento

```
*Main> sumador 1 (1/0)
2
*Main>
```

Haskell: evaluación de funciones

- Ejemplo: cálculo del cuadrado de un número

```
-- sq x devuelve el cuadrado x
sq :: Integer -> Integer
sq x = x * x
```

Haskell: evaluación de funciones

- ¿Cómo se evalúa sq 5?
  - Usando la definición
    - Subtituyendo 5 por x
    - *sq 5 = 5 * 5*
  - Continuar evaluando las expresiones
    - sq 5 = 25
- Funciona como en matemática
- Importante: No existe estado

Haskell

- Tipos de Datos:
  - Int, Float, Bool, Char, etc.
- Ejemplos de Caracteres
- String
- **putStr** vs **print**

Haskell

- Tuplas
  - Mostrar ejemplo
  - Consideraciones
- Alcances
  - Ejemplo
  - Traza

haskell4.hs

```haskell
parEnteros :: (Int, Int)
parEnteros = (32, 33)

sumapar :: (Int, Int) -> Int
sumapar (x, y) = x + y
```

```
*Main> :l haskell4.hs
[1 of 1] Compiling Main             ( haskell4.hs, interpreted )
Ok, modules loaded: Main.
*Main> parEnteros
(32,33)
*Main> sumapar parEnteros
65
*Main> sumapar (10,10)
20
*Main> sumapar 10 10

<interactive>:24:1:
    Couldn't match expected type `a0 -> t0' with actual type `Int'
    The function `sumapar' is applied to two arguments,
    but its type `(Int, Int) -> Int' has only one
    In the expression: sumapar 10 10
    In an equation for `it': it = sumapar 10 10
*Main>
```

haskell4.hs

```haskell
sumaCuadrados :: Int -> Int -> Int
sumaCuadrados n m
  = cuadrN + cuadrM
  where
    cuadrN = n*n
    cuadrM = m*m

-- sumaCuadrados 4 3
-- = cuadrN + cuadrM
--     where
--         cuadrN = 4*4 = 16
--         cuadrM = 3*3 = 9
-- = 16 + 9
-- = 25
```

Lenguajes Funcionales

- Haskell
  - Lenguaje Funcional Puro y Moderno
    - Sin variables, sin efectos colaterales, sin asignaciones
  - Estático, fuertemente tipado, sobrecargado
  - Perezoso
    - No evalúa una (sub)expresión sino hasta que la necesita
    - Define listas por comprensión, lo que le permite manipular estructuras infinitas
  - Currificado
    - Una función con muchos parámetros es currificada si se puede ver como una función de algo nivel de un parámetro único
  - Otras características únicas

Apunte Lenguaje Haskell

- Disponible en el sitio web de la asignatura
- Se presenta sólo la primera parte.
- Los ejemplos pueden seguirse directamente junto con el apunte.