Fundamentos de Programación Funcional y Concurrente

Principios de Programación Funcional

Juan Francisco Díaz Frias

Profesor Titular (1993-hoy)
juanfco.diaz@correounivalle.edu.co
Edif. B13 - 4009
Universidad del Valle

Septiembre 2022

Plan

Plan

1. Principios
   - Paradigmas de programación
   - ¿Cómo escalar la programación?
   - Principios de la PF
2. Fundamentos
   - Los conceptos de función y de composición
   - Cómo se definen funciones
3. Lenguajes de Programación Funcional
   - Lenguajes de Programación Funcional
   - Scala: El LP del curso

Plan

1. Principios
   - Paradigmas de programación
   - ¿Cómo escalar la programación?
   - Principios de la PF
2. Fundamentos
   - Los conceptos de función y de composición
   - Cómo se definen funciones
3. Lenguajes de Programación Funcional
   - Lenguajes de Programación Funcional
   - Scala: El LP del curso

Plan

Paradigmas de programación

- Un paradigma describe distintos conceptos y patrones de pensamiento en una disciplina científica.
- Principales paradigmas de programación:
  - Declarativo (Funcional, Lógico, Descriptivo)
  - Concurrente (Declarativo, por paso de mensajes, con estado)
  - Con estado (conocido como imperativo)
  - Orientado a Objetos (no es más que el paradigma con estado)
  - Relacional
  - Distribuido
  - ...

Paradigmas de programación

- Un paradigma describe distintos conceptos y patrones de pensamiento en una disciplina científica.
- Principales paradigmas de programación:
  - Declarativo (Funcional, Lógico, Descriptivo)
  - Concurrente (Declarativo, por paso de mensajes, con estado)
  - Con estado (conocido como imperativo)
  - Orientado a Objetos (no es más que el paradigma con estado)
  - Relacional
  - Distribuido
  - ...

El paradigma imperativo

- La programación imperativa trata sobre:
  - Variables mutables (conocidas como celdas)
  - El uso de la asignación
  - El uso de estructuras de control (if-then-else, ciclos for o while, breaks, continue, return)
- Es una abstracción de las máquinas de Von Newman. Existe una correspondencia entre
  - Variables mutables $\approx$ celdas de memoria
  - Desreferenciación de Variables $\approx$ instrucciones de carga
  - Asignación de Variables $\approx$ instrucciones de almacenamiento
  - Estructuras de control $\approx$ saltos (jumps)
- Reto: Escalar la programación ¿Cómo conceptualizar los programas más allá de *instrucción por instrucción* sobre datos simples?
- Necesitamos otras técnicas para definir altos niveles de abstracción (colecciones, polinomios, formas geométricas, cadenas, documentos, ...): Teorías

El paradigma imperativo

- La programación imperativa trata sobre:
  - Variables mutables (conocidas como celdas)
  - El uso de la asignación
  - El uso de estructuras de control (if-then-else, ciclos for o while, breaks, continue, return)
- Es una abstracción de las máquinas de Von Newman. Existe una correspondencia entre
  - Variables mutables $\approx$ celdas de memoria
  - Desreferenciación de Variables $\approx$ instrucciones de carga
  - Asignación de Variables $\approx$ instrucciones de almacenamiento
  - Estructuras de control $\approx$ saltos (jumps)
- Reto: Escalar la programación ¿Cómo conceptualizar los programas más allá de *instrucción por instrucción* sobre datos simples?
- Necesitamos otras técnicas para definir altos niveles de abstracción (colecciones, polinomios, formas geométricas, cadenas, documentos, ...): Teorías

El paradigma imperativo

- La programación imperativa trata sobre:
  - Variables mutables (conocidas como celdas)
  - El uso de la asignación
  - El uso de estructuras de control (if-then-else, ciclos for o while, breaks, continue, return)
- Es una abstracción de las máquinas de Von Newman. Existe una correspondencia entre
  - Variables mutables $\approx$ celdas de memoria
  - Desreferenciación de Variables $\approx$ instrucciones de carga
  - Asignación de Variables $\approx$ instrucciones de almacenamiento
  - Estructuras de control $\approx$ saltos (jumps)
- Reto: Escalar la programación ¿Cómo conceptualizar los programas más allá de *instrucción por instrucción* sobre datos simples?
- Necesitamos otras técnicas para definir altos niveles de abstracción (colecciones, polinomios, formas geométricas, cadenas, documentos, ...): Teorías

El paradigma imperativo

- La programación imperativa trata sobre:
  - Variables mutables (conocidas como celdas)
  - El uso de la asignación
  - El uso de estructuras de control (if-then-else, ciclos for o while, breaks, continue, return)
- Es una abstracción de las máquinas de Von Newman. Existe una correspondencia entre
  - Variables mutables $\approx$ celdas de memoria
  - Desreferenciación de Variables $\approx$ instrucciones de carga
  - Asignación de Variables $\approx$ instrucciones de almacenamiento
  - Estructuras de control $\approx$ saltos (jumps)
- Reto: Escalar la programación ¿Cómo conceptualizar los programas más allá de *instrucción por instrucción* sobre datos simples?
- Necesitamos otras técnicas para definir altos niveles de abstracción (colecciones, polinomios, formas geométricas, cadenas, documentos, ...): Teorías

Plan

Teorías

Una teoría consiste de:

- Uno o más tipos de datos
- Operaciones sobre esos tipos
- Leyes que describan las relaciones entre valores y operaciones

Normalmente, una teoría no describe mutaciones

Teorías

Una teoría consiste de:

- Uno o más tipos de datos
- Operaciones sobre esos tipos
- Leyes que describan las relaciones entre valores y operaciones

Normalmente, una teoría no describe mutaciones
Por ejemplo, la teoría de cadenas, define un operador $++$ que es asociativo:

$$(a ++ b) ++ c = a ++ (b ++ c)$$

pero no define un operador para cambiarle un elemento a la cadena y que ella siga siendo la misma

Consecuencias para la programación

Al programar

- Nos concentraremos en definir teorías para operadores
- Minimizaremos los cambios de estado
- Trataremos los operadores como funciones, muchas veces compuestas por funciones más sencillas

Consecuencias para la programación

Al programar

- Nos concentraremos en definir teorías para operadores
- Minimizaremos los cambios de estado
- Trataremos los operadores como funciones, muchas veces compuestas por funciones más sencillas

Consecuencias para la programación

Al programar

- Nos concentraremos en definir teorías para operadores
- Minimizaremos los cambios de estado
- Trataremos los operadores como funciones, muchas veces compuestas por funciones más sencillas

Plan

Principios de la PF

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, la PF significa que un programa es un conjunto de funciones e invocaciones a funciones.
- En particular, las funciones son valores que pueden ser producidos, consumidos y compuestos.
- Los lenguajes de programación funcionales hacen que esto sea sencillo.

Principios de la PF

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, la PF significa que un programa es un conjunto de funciones e invocaciones a funciones.
- En particular, las funciones son valores que pueden ser producidos, consumidos y compuestos.
- Los lenguajes de programación funcionales hacen que esto sea sencillo.

Principios de la PF

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, la PF significa que un programa es un conjunto de funciones e invocaciones a funciones.
- En particular, las funciones son valores que pueden ser producidos, consumidos y compuestos.
- Los lenguajes de programación funcionales hacen que esto sea sencillo.

Principios de la PF

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, la PF significa que un programa es un conjunto de funciones e invocaciones a funciones.
- En particular, las funciones son valores que pueden ser producidos, consumidos y compuestos.
- Los lenguajes de programación funcionales hacen que esto sea sencillo.

Principios de los lenguajes de PF

- En un sentido operativo, un lenguaje de PF no ofrece variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, un lenguaje de PF permite la construcción de programas elegantes enfocados en funciones.
- En particular, las funciones en un lenguaje de PF son *ciudadanos de primera clase*, es decir:
  - Pueden ser definidas en cualquier parte, incluso dentro de otras funciones.
  - Al igual que cualquier otro valor, las funciones pueden ser pasadas como parámetros a otras funciones y devueltas como resultado de aplicar una función.
  - Como para cualquier valor, existen operadores sobre las funciones (para componerlas y para invocarlas).

Principios de los lenguajes de PF

- En un sentido operativo, un lenguaje de PF no ofrece variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, un lenguaje de PF permite la construcción de programas elegantes enfocados en funciones.
- En particular, las funciones en un lenguaje de PF son *ciudadanos de primera clase*, es decir:
  - Pueden ser definidas en cualquier parte, incluso dentro de otras funciones.
  - Al igual que cualquier otro valor, las funciones pueden ser pasadas como parámetros a otras funciones y devueltas como resultado de aplicar una función.
  - Como para cualquier valor, existen operadores sobre las funciones (para componerlas y para invocarlas).

Principios de los lenguajes de PF

- En un sentido operativo, un lenguaje de PF no ofrece variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, un lenguaje de PF permite la construcción de programas elegantes enfocados en funciones.
- En particular, las funciones en un lenguaje de PF son *ciudadanos de primera clase*, es decir:
  - Pueden ser definidas en cualquier parte, incluso dentro de otras funciones.
  - Al igual que cualquier otro valor, las funciones pueden ser pasadas como parámetros a otras funciones y devueltas como resultado de aplicar una función.
  - Como para cualquier valor, existen operadores sobre las funciones (para componerlas y para invocarlas).

Plan

Los conceptos de función y de composición

- Las funciones

$$\begin{array}{ccc} \mathit{Dominio} & \xrightarrow{f} & \mathit{Rango} \\ x \in \mathit{Dominio} & \mapsto & f(x) \in \mathit{Rango} \end{array}$$

- La composición de funciones

$$\begin{array}{ccccc} A & \xrightarrow{g} & B & \xrightarrow{f} & C \\ x & \mapsto & g(x) & \mapsto & f(g(x)) \end{array}$$

- Destacar:
  - Noción de tipo
  - Noción de abstracción funcional: $h(x) = f(g(x))$

Los conceptos de función y de composición

- Las funciones

$$\begin{array}{ccc} \mathit{Dominio} & \xrightarrow{f} & \mathit{Rango} \\ x \in \mathit{Dominio} & \mapsto & f(x) \in \mathit{Rango} \end{array}$$

- La composición de funciones

$$\begin{array}{ccccc} A & \xrightarrow{g} & B & \xrightarrow{f} & C \\ x & \mapsto & g(x) & \mapsto & f(g(x)) \end{array}$$

- Destacar:
  - Noción de tipo
  - Noción de abstracción funcional: $h(x) = f(g(x))$

Los conceptos de función y de composición

- Las funciones

$$\begin{array}{ccc} \mathit{Dominio} & \xrightarrow{f} & \mathit{Rango} \\ x \in \mathit{Dominio} & \mapsto & f(x) \in \mathit{Rango} \end{array}$$

- La composición de funciones

$$\begin{array}{ccccc} A & \xrightarrow{g} & B & \xrightarrow{f} & C \\ x & \mapsto & g(x) & \mapsto & f(g(x)) \end{array}$$

- Destacar:
  - Noción de tipo
  - Noción de abstracción funcional: $h(x) = f(g(x))$

Plan

Cómo se definen funciones

- Tabulación o enumeración:

x	a	b	c
f(x)	1	1	2

- Reglas conocidas

$$Suc(x) = x + 1, Cubo(y) = y \times y \times y$$

- Composición

$$h(x) = Suc(Cubo(x)) = x^3 + 1$$

- Casos

$$max(x, y) = \begin{cases} x & \text{Si } x \geq y \\ y & \text{En caso contrario} \end{cases}$$

- Recursividad

$$0! = 1, n! = n \times (n-1)! \text{Si } n > 0$$

Cómo se definen funciones

- Tabulación o enumeración:

x	a	b	c
f(x)	1	1	2

- Reglas conocidas

$$Suc(x) = x + 1, Cubo(y) = y \times y \times y$$

- Composición

$$h(x) = Suc(Cubo(x)) = x^3 + 1$$

- Casos

$$max(x, y) = \begin{cases} x & \text{Si } x \geq y \\ y & \text{En caso contrario} \end{cases}$$

- Recursividad

$$0! = 1, n! = n \times (n-1)! \text{Si } n > 0$$

Cómo se definen funciones

- Tabulación o enumeración:

x	a	b	c
f(x)	1	1	2

- Reglas conocidas

$$Suc(x) = x + 1, Cubo(y) = y \times y \times y$$

- Composición

$$h(x) = Suc(Cubo(x)) = x^3 + 1$$

- Casos

$$max(x, y) = \begin{cases} x & \text{Si } x \geq y \\ y & \text{En caso contrario} \end{cases}$$

- Recursividad

$$0! = 1, n! = n \times (n-1)! \text{Si } n > 0$$

Cómo se definen funciones

- Tabulación o enumeración:

x	a	b	c
f(x)	1	1	2

- Reglas conocidas

$$Suc(x) = x + 1, Cubo(y) = y \times y \times y$$

- Composición

$$h(x) = Suc(Cubo(x)) = x^3 + 1$$

- Casos

$$max(x, y) = \begin{cases} x & \text{Si } x \geq y \\ y & \text{En caso contrario} \end{cases}$$

- Recursividad

$$0! = 1, n! = n \times (n-1)! \text{Si } n > 0$$

Cómo se definen funciones

- Tabulación o enumeración:

x	a	b	c
f(x)	1	1	2

- Reglas conocidas

$$Suc(x) = x + 1, Cubo(y) = y \times y \times y$$

- Composición

$$h(x) = Suc(Cubo(x)) = x^3 + 1$$

- Casos

$$max(x, y) = \begin{cases} x & \text{Si } x \geq y \\ y & \text{En caso contrario} \end{cases}$$

- Recursividad

$$0! = 1, n! = n \times (n-1)! \text{Si } n > 0$$

Plan

Lenguajes de Programación Funcional

En un sentido operativo:

- Pure Lisp, XSLT, SPath, XQuery, FP
- Haskell (sin I/O Monad o UnsafePerformIO)

Lenguajes de Programación Funcional

En un sentido operativo:

- Pure Lisp, XSLT, SPath, XQuery, FP
- Haskell (sin I/O Monad o UnsafePerformIO)

En un sentido amplio:

- Lisp,Scheme, Racket, Clojure
- SML, CAML, OCAML, F#
- Haskell (Full languahe)
- Scala
- Smalltalk, Ruby

Plan

Scala: El LP del curso

Usaremos Scala como el lenguaje de programación del curso.
¿Por qué Scala?

- Porque corre sobre JVM y su sintaxis es muy parecida a la de Java (menos esfuerzo inicial)
- Porque implementa el paradigma funcional de forma elegante
- Porque implementa la concurrencia de alto nivel (más allá de la concurrencia con estado compartido)
- Porque ha sido diseñado para el procesamiento de grandes cantidades de datos, y eso puede ser muy útil para los estudiantes.

Scala: El LP del curso

Usaremos Scala como el lenguaje de programación del curso.
¿Por qué Scala?

- Porque corre sobre JVM y su sintaxis es muy parecida a la de Java (menos esfuerzo inicial)
- Porque implementa el paradigma funcional de forma elegante
- Porque implementa la concurrencia de alto nivel (más allá de la concurrencia con estado compartido)
- Porque ha sido diseñado para el procesamiento de grandes cantidades de datos, y eso puede ser muy útil para los estudiantes.

Scala: El LP del curso

Usaremos Scala como el lenguaje de programación del curso.
¿Por qué Scala?

- Porque corre sobre JVM y su sintaxis es muy parecida a la de Java (menos esfuerzo inicial)
- Porque implementa el paradigma funcional de forma elegante
- Porque implementa la concurrencia de alto nivel (más allá de la concurrencia con estado compartido)
- Porque ha sido diseñado para el procesamiento de grandes cantidades de datos, y eso puede ser muy útil para los estudiantes.

Scala: El LP del curso

Usaremos Scala como el lenguaje de programación del curso.
¿Por qué Scala?

- Porque corre sobre JVM y su sintaxis es muy parecida a la de Java (menos esfuerzo inicial)
- Porque implementa el paradigma funcional de forma elegante
- Porque implementa la concurrencia de alto nivel (más allá de la concurrencia con estado compartido)
- Porque ha sido diseñado para el procesamiento de grandes cantidades de datos, y eso puede ser muy útil para los estudiantes.