Taller 2: Funciones de alto orden
Fundamentos de Programación Funcional y Concurrente

Carlos Andres Delgado S
carlos.andres.delgado@correounivalle.edu.co

Septiembre de 2024

1. Ejercicios de programación: Conjuntos difusos

En informática, el concepto de conjunto es fundamental para modelar cualquier problema. Por ejemplo, podemos hablar del conjunto de los números pares, de los múltiplos de 5 o del conjunto de carros con placa terminada en 7, entre muchos otros conjuntos definidos precisamente. La característica principal de los conjuntos mencionados es que, dado un elemento del universo que conforma los conjuntos, es preciso decidir si el elemento pertenece o no al conjunto. Una forma de modelar los conjuntos es por medio de una función denominada la **función característica**. Dado un universo U y $S \subseteq U$, la función característica de S es una función $f_S : U \to \{0, 1\}$ tal que:

- $\forall s \in S : f_S(s) = 1$
- $\forall s \notin S : f_S(s) = 0$

Sin embargo, en el mundo real también hay ejemplos de conjuntos en los que definir si un elemento pertenece o no al conjunto no es tan preciso. Por ejemplo, el conjunto de los hombres bajitos o el conjunto de los hombres altos. Seguramente hay personas que uno puede clasificar claramente como bajitas, otras como altas, pero también hay otras que, dependiendo de la perspectiva, podrían caer en un conjunto o en el otro. Lo mismo sucede, por ejemplo, con el conjunto de los carros rojos. Habrá carros que uno claramente clasifica como rojos, otros como no rojos, pero algunos pueden clasificarse como rojos dependiendo del matiz.

Para modelar este tipo de realidades, Lofti Zadeh introdujo en 1965 la definición de **conjuntos difusos**. La gran diferencia es que la función característica de un conjunto difuso es ahora una función que define el grado de pertenencia de un elemento al conjunto, en lugar de definir solamente si pertenece o no. Dado un universo U y $S \subseteq U$, la función característica de S es una función $f_S : U \to [0, 1]$ tal que $\forall s \in U : f_S(s) \in [0, 1]$. Ahora:

- Si $f_S(s) = 0$, eso significa que $s \notin S$.
- Si $f_S(s) = 1$, eso significa que $s \in S$.
- Si $f_S(s) = 0{,}2$, no podemos decir categóricamente que $s \notin S$, aunque tiene pocas características que lo clasifiquen allí.

- Si $f_S(s) = 0{,}8$, no podemos decir categóricamente que $s \in S$, aunque tiene muchas características que lo clasifican allí.

En este taller, trabajaremos con una representación funcional de conjuntos difusos basada en la noción de función característica propuesta por Zadeh.

1.1. Funciones básicas sobre conjuntos difusos

Los conjuntos que representaremos serán conjuntos difusos de números enteros. En consecuencia, vamos a definir el tipo `ConjDifuso` en Scala de la siguiente manera:

```
type ConjDifuso = Int => Double
```

Usando esta representación, podemos definir la función que verifica el grado de pertenencia de un elemento a un conjunto difuso de la siguiente manera:

```
def pertenece(elem: Int, s: ConjDifuso): Double = {
  s(elem)
}
```

Por ejemplo, si se desea representar el conjunto de números enteros mucho mayores que 1, podríamos definir una función característica que:

- Para todos los números menores o iguales a 1, dé 0.
- Para todos los números mayores o iguales a 3, dé 1 (considerando que los números a partir de 3 son mucho mayores que 1).
- Para los números entre 1 y 3, dé un número entre 0 y 1, que crece proporcionalmente a medida que se aleja de 1 y se acerca a 3.

La función `muchoMayorQue` definida en Scala devuelve el conjunto difuso que representa los números enteros mucho mayores que a, siguiendo la idea mencionada anteriormente:

```
def muchoMayorQue(a: Int, m: Int): ConjDifuso = {
  def mma(x: Int): Double = {
    if (x <= a) 0.0
    else if (x > a && x <= m) (x - a).toDouble / (m - a).toDouble
    else 1.0
  }
  mma
}
```

Así, `mm1` y `mm2` en el siguiente programa representan los conjuntos difusos de números mucho mayores que 1 y mucho mayores que 2, respectivamente:

```
val mm1 = muchoMayorQue(1, 3)
val mm2 = muchoMayorQue(2, 6)
```

1.2. Funciones básicas sobre conjuntos difusos

Su tarea es implementar las siguientes funciones básicas sobre conjuntos difusos:

1.2.1. Conjunto difuso de números grandes

¿Cómo distinguir si un número entero n es grande? Una idea es calcular $\frac{n}{n+d}$, donde d es un número pequeño mayor o igual a 1. Si n es grande, $n+d$ está muy cerca de n, y esa división da un valor cercano a 1. Para mejorar la calidad del grado de pertenencia, se podría calcular $\left(\frac{n}{n+d}\right)^e$, donde e es un entero mayor que 1.

Defina una función `grande`, que recibe un número entero d y un número entero e, y crea el conjunto difuso de números grandes:

```
def grande(d: Int, e: Int): ConjDifuso = {
  ...
}
```

Ahora que podemos crear algunos conjuntos difusos, vamos a crear conjuntos difusos nuevos a partir de ellos, uniendo, intersectando y haciendo complementos de conjuntos difusos.

1.2.2. Complemento, Unión e Intersección

Defina la función `complemento`, que recibe un conjunto difuso de enteros y devuelve el conjunto difuso correspondiente a su complemento. Nótese que si f_S es la función característica del conjunto difuso $S \subseteq U$, entonces $f_{\neg S} = 1 - f_S$.

```
def complemento(c: ConjDifuso): ConjDifuso = {
  ...
}
```

Defina las funciones `union` e `interseccion`, que reciben dos conjuntos difusos de enteros y devuelven el conjunto difuso correspondiente a la unión y a la intersección, respectivamente, de esos dos conjuntos. Nótese que si f_{S_1} y f_{S_2} son las funciones características de los conjuntos difusos $S_1, S_2 \subseteq U$, entonces:

$$f_{S_1 \cup S_2} = \text{máx}(f_{S_1}, f_{S_2})$$
$$f_{S_1 \cap S_2} = \text{mín}(f_{S_1}, f_{S_2})$$

```
def union(cd1: ConjDifuso, cd2: ConjDifuso): ConjDifuso = {
  ...
}

def interseccion(cd1: ConjDifuso, cd2: ConjDifuso): ConjDifuso = {
  ...
}
```

1.2.3. Inclusión e igualdad

En los conjuntos clásicos, se tienen dos relaciones fundamentales: la inclusión ($\subseteq$) y la igualdad ($=$) de conjuntos. ¿Cómo se extienden estas nociones a conjuntos difusos?

Dados S_1, S_2 dos conjuntos difusos, se dice que $S_1 \subseteq S_2$ si el grado de pertenencia de cada elemento a S_1 es menor o igual al grado de pertenencia de ese mismo elemento a S_2, es decir, si $\forall s \in U : f_{S_1}(s) \leq f_{S_2}(s)$.

Dados S_1, S_2 dos conjuntos difusos, se dice que $S_1 = S_2$ si $S_1 \subseteq S_2 \wedge S_2 \subseteq S_1$.

Defina las funciones `inclusion` e `igualdad`, que reciben dos conjuntos difusos y devuelven `true` si el primero está incluido o es igual al segundo, respectivamente. En caso contrario, devuelven `false`. Nótese que no existe una manera directa de listar todos los elementos de un conjunto. La función `pertenece` sólo permite verificar en qué grado un entero forma parte de un conjunto o no. Por tanto, si se quiere iterar sobre todos los elementos de un conjunto, hay que asumir que los enteros están en el intervalo $[0, 1000]$ para limitar el espacio de búsqueda.

Implemente `inclusion` usando recursión de cola. Use una función auxiliar interna.

```
def inclusion(cd1: ConjDifuso, cd2: ConjDifuso): Boolean = {
  ...
}

def igualdad(cd1: ConjDifuso, cd2: ConjDifuso): Boolean = {
  ...
}
```

2. Entructura entregable

Usted deberá realizar su solución como clase en el paquete taller o creando alguno de su preferencia, la estructura para la entrega es:

```
class ConjuntosDifusos {

  type ConjDifuso = Int => Double

  def pertenece(elem: Int, s: ConjDifuso): Double = {
    s(elem)
  }

  def grande(d: Int, e: Int): ConjDifuso = {
    // Implementación de la función grande
    ...
  }

  def complemento(c: ConjDifuso): ConjDifuso = {
    // Implementación de la función complemento
    ...
  }

  def union(cd1: ConjDifuso, cd2: ConjDifuso): ConjDifuso = {
    // Implementación de la función union
    ...
  }

  def interseccion(cd1: ConjDifuso, cd2: ConjDifuso): ConjDifuso = {
    // Implementación de la función interseccion
    ...
```

```
  }

  def inclusion(cd1: ConjDifuso, cd2: ConjDifuso): Boolean = {
    // Implementación de la función inclusion
    ...
  }

  def igualdad(cd1: ConjDifuso, cd2: ConjDifuso): Boolean = {
    // Implementación de la función igualdad
    ...
  }

}
```

2.1. Informe del taller - secciones

Todo taller debe venir acompañado de un informe en formato PDF. El informe debe contener la información explícita solicitada en el enunciado.

Para este caso, el informe del taller debe contener al menos tres secciones: informe de procesos, informe de corrección y conclusiones.

Use un formato sencillo, no olvide colocar los nombres completos y códigos de todos los integrantes del grupo.

2.1.1. Informe de procesos

Tal como se ha visto en clase, los procesos generados por los programas recursivos, genere un ejemplo para cada ejercicio, y muestre cómo se comporta el proceso generado por cada uno de los programas. Para ello, debe mostrar la pila de llamadas que se genera en cada caso, y cómo se va desplegando la pila de llamadas a medida que se resuelve el problema. Use ejemplos con valores pequeños.

2.1.2. Informe de corrección

Es muy importante reflexionar sobre la corrección del código entregado. Para ello se deberá argumentar sobre la corrección de los programas entregados, y también deberá entregar un conjunto de pruebas. Todo esto lo consigna en esta sección del informe, dividida de la siguiente manera:

Argumentación sobre la corrección Para cada función, `pertenece`, `grande`, `complemento`, `union`, `interseccion`, `inclusion` e `igualdad`, argumente si la implementación es correcta o no. Use notación matemática para argumentar la corrección de los algoritmos.

Casos de prueba Para cada función se requieren mínimo 5 casos de prueba donde se conozca claramente el valor esperado, y se pueda evidenciar que el valor calculado por la función corresponde con el valor esperado en toda ocasión. Agreguelos como pruebas de software. Estas pruebas debe ser disponibles en el paquete taller, de test.

2.2. Condiciones de entrega

La fecha de entrega máxima es viernes 04 de Octubre a las 23:59:59, se aplicará una penalización de 0.15 por cada hora o fracción de retraso. Por ejemplo, si se entrega a partir a las 00:00:01, la nota máxima será de 4.85, si se entrega a partir las 01:00:00, la nota máxima será de 4.70, y así sucesivamente. La inscripción de grupos estará disponible hasta el 01 de Octubre de 2024 a las 23:59:59.

Las condiciones de entrega son:

1. Los estudiantes deben inscribirse a un grupo en el campus virtual, no hacerlo implica 0.0 en la nota del taller.
2. Debe hacerse un fork del repositorio: `https://github.com/cardel/talleres-pfc-template`, recuerde asignar un nombre diferente en su cuenta, así mismo agregar como colaboradores a sus compañeros.
3. Debe entregar el enlace de su repositorio en el Campus Virtual antes del cierre indicado por el docente
4. Las fecha del último commit debe ser antes del cierre indicado por el docente
5. Incluya el informe en la raíz del repositorio en formato PDF
6. El docente verificará con herramientas antiplagio la originalidad de los talleres entregados, así mismo se tomará en cuenta la interacción de commits dentro del repositorio de github y su trazabilidad (flujo de trabajo).
7. El docente puede requerir que se sustente el taller, la nota del taller será individual y será entre 0 y 1, la cual se multiplica por la nota grupal. Por ejemplo, si usted obtiene 0.5 y la nota de su grupo es 5.0, su nota de taller será 2.5.

3. Rubrica de evaluación

3.1. Regla del taller

Criterio	**No cumple (0 puntos)**	**Cumple (10 puntos)**
Informe en formato PDF y disponible en la raíz del proyecto		
El proyecto es un fork del indicado por el docente		
El informe tiene los nombres completos y códigos de los estudiantes		

En total se pueden obtener 30 puntos en la regla del taller.

3.2. Rubrica de evaluación puntos

Importante El taller debe entregarse en su totalidad, en caso de que no se entregue algún punto o este no es funcional, máximo se asignará nivel 1 en la rubrica de evaluación.

Criterio	**Nivel 0 (0 puntos)**	**Nivel 1 (5 puntos)**	**Nivel 2 (10 puntos)**	**Nivel 3 (15 puntos)**
Solución del problema	No se entrega o es no funcional.	La implementación no es correcta y produce resultados erróneos	La implementación es correcta pero no se sigue el enfoque de conjuntos como funciones indicadas en el enunciado	La implementación es correcta y sigue el enfoque de conjuntos como funciones como se indica en el enunciado
Informe de proceso	No se entrega o es no funcional.	No se explica correctamente el enfoque de funciones de alto orden en la solución del problema	Se explica el enfoque de funciones de alto orden en la solución del problema, pero la explicación carece de profundidad y análisis de la solución presentada	Se explica el enfoque de funciones de alto orden en la solución del problema, es profunda, utiliza los conceptos vistos en clase y se analiza correctamente la solución presentada
Informe de corrección	No se entrega o es no funcional.	Argumenta la corrección de los algoritmos de forma parcial; la notación matemática tiene errores menores.	Argumenta claramente la corrección de los algoritmos; la notación matemática es en su mayoría correcta.	Argumenta claramente utilizando notación matemática para demostrar que los algoritmos implementados son correctos; muestra cómo son los llamados. La notación es correcta y clara.

Criterio	**Nivel 0 (0 puntos)**	**Nivel 1 (5 puntos)**	**Nivel 2 (10 puntos)**	**Nivel 3 (15 puntos)**
Casos de prueba*	No se entrega o es no funcional.	Incluye algunos casos de prueba, pero no son significativos o no se ejecutan al construir el proyecto.	Incluye 5 ejemplos para algunos puntos del taller; se ejecutan al construir el proyecto y son casos mayormente significativos.	Incluye en el código 5 ejemplos para cada punto del taller como pruebas de software; estas se ejecutan al construir el proyecto y son casos significativos (no triviales) de la solución del problema.

* Los casos de prueba deben ser pruebas de software dentro de la carpeta test, estas se deben ejecutar dentro de las rutinas de Gradle.

En total se pueden obtener 60 puntos en la rubrica de evaluación.

3.3. Calificación

La calificación final del taller será la suma de los puntos obtenidos en la regla del taller y la rubrica de evaluación. La calificación máxima es de 90 puntos.

La nota grupal N, se pondera entre 0 y 5, de acuerdo a los puntos P obtenidos de la siguiente manera:

$$\nu = \frac{P}{90} \times 5$$

Tener en cuenta que esta es la nota grupal, y que la nota individual en caso de requerir sustentación se multiplica por esta nota grupal.