Práctica 6:
Gestión de la incertidumbre

Introducción y Objetivos

El objetivo de la práctica es el uso de un software para trabajar con sistemas basados en reglas difusas. Concretamente, se pretende:

- Modelar una base de conocimiento compuesta por variables lingüísticas, etiquetas lingüísticas y reglas difusas.
- Ejecutar inferencias sobre la base de conocimiento, calculando valores de salida para unos valores de entrada dados.
- Conocer algunas de las opciones disponibles en cuanto a tipo de reglas, operadores de unión, métodos de defuzzificación, etc.

Como aplicación concreta, se propone resolver el problema del péndulo invertido. Como siempre, esta aplicación concreta podría adaptarse según los intereses del estudiante.

JFML

JFML (http://www.uco.es/JFML) es una biblioteca Java para manejar sistemas basados en reglas difusas desarrollada por investigadores españoles e italianos. La herramienta se puede descargar gratuitamente del repositorio GitHub (https://github.com/sotillo19/JFML). En la carpeta *"Examples"* hay ficheros .jar que podemos incluir en nuestro proyecto Java; recomendamos usar la versión 1.2.2.

Puede encontrarse más información sobre JFML en la página web de la herramienta, incluyendo un manual de usuario, documentación generada con Javadoc, o un artículo científico de acceso abierto (http://doi.org/10.1109/ACCESS.2018.2872777).

Para representar un sistema basado en reglas difusas, la clase principal es `FuzzyInferenceSystem`, que representa un sistema de inferencia y que tendrá asociada una base de conocimiento (`KnowledgeBaseType`) y una base de reglas (clase `FuzzyRuleType`).

La base de conocimiento se compone de variables lingüísticas (clase `FuzzyVariableType`). Para toda variable lingüística se debe especificar:

- un dominio,
- un valor por defecto y
- un conjunto de etiquetas lingüísticas (clase `FuzzyTermType`).

Para crear las etiquetas lingüísticas, existen varias funciones predefinidas, como por ejemplo:

- L: `FuzzyTermType.TYPE_leftLinearShape`
- Triangular: `FuzzyTermType.TYPE_triangularShape`
- Trapezoidal: `FuzzyTermType.TYPE_trapezoidShape`
- Γ: `FuzzyTermType.TYPE_rightLinearShape`

Para las variables de salida, además, se debe especificar:

- un mecanismo de agregación de las contribuciones de cada regla, como por ejemplo el máximo (`"MAX"`) o la suma probabilística (`"PROBOR"`), y
- un método de defuzzificación, como el centro de graves (`"COG"`) o el punto medio de los máximos (`"MOM"`).

Para crear la base de reglas, existen varias subclases de `FuzzyRuleType`, como por ejemplo `MamdaniRuleType`. Para crear una regla, podemos usar el método constructor

```
FuzzyRuleType(java.lang.String name, java.lang.String connector,
              java.lang.String connectorMethod, java.lang.Float weight)
```

que recibe los siguientes parámetros:

- `name`: Nombre de la regla,
- `connector`: Tipo de conector, conjuntivo (`"and"`) o disyuntivo (`"or"`),
- `connectorMethod`: Función asociada al conector, como `"MIN"` o `"MAX"` y
- `weight`: importancia relativa de la regla medida en [0,1].

Cada regla tiene un antecedente (`AntecedentType`) y un consecuente (`ConsequentType`) donde aparecerán cláusulas sobre los valores de variables lingüísticas (clase `FuzzyVariableType`), definidas mediante etiquetas lingüísticas (clase `FuzzyTermType`). El método `addClause` permite asociar una cláusula a un antecedente o consecuente de una regla:

```
addClause(KnowledgeBaseVariable variable, FuzzyTerm term)
```

siendo KnowledgeBaseVariable y FuzzyTerm superclass de FuzzyVariableTypey FuzzyTermType.

Ejercicios

1. Crear una base de conocimiento y una base de reglas para resolver el problema del péndulo invertido. Suponga la definición de las etiquetas lingüísticas y las reglas mostradas en las transparencias de teoría.

2. Construir una aplicación Java que pida al usuario los valores de las variables de entrada (ángulo y velocidad angular) y que muestre el valor de la variable de salida (velocidad de la plataforma).

3. Extender la aplicación para que permita elegir entre dos métodos de agregación (`"MAX"`o `"PROBOR"`) y entre dos métodos de defuzzificación (`"COG"` y `"MOM"`), observando las diferencias de funcionamiento.

Entrega

A través de Moodle hasta las 11 horas del **7 de mayo**. Será un fichero `.zip` que incluya al menos

- la carpeta `src` del proyecto,
- un ejecutable `.jar` y
- una brevísima documentación en `pdf` (autores, aspectos no triviales …)

Apéndice. Plantilla para JFML

```
import jfml.*;
import jfml.knowledgebase.*;
import jfml.knowledgebase.variable.*;
import jfml.rule.*;
import jfml.rulebase.*;
import jfml.term.*;

public class Practica6 {

   public static void main(String[] args) {

      FuzzyInferenceSystem fis = new FuzzyInferenceSystem("Péndulo invertido");
      KnowledgeBaseType kb = new KnowledgeBaseType();
      fis.setKnowledgeBase(kb);

      // Variable de entrada: ángulo
      FuzzyVariableType angulo = new FuzzyVariableType("Ángulo", -180, 180);
      angulo.setType("input");
      kb.addVariable(angulo);

      // Términos lingüísticos de Ángulo
      FuzzyTermType angNG = new FuzzyTermType("Negativo Grande",
            FuzzyTermType.TYPE_leftLinearShape, (new float[] { -30f, -15f }));
      angulo.addFuzzyTerm(angNG);
      // Añadir otros términos lingüísticos ...
      // añadir otras variables de entrada ...

      // Variable de salida: velocidad de la plataforma
      FuzzyVariableType velocidad = new FuzzyVariableType("Velocidad", -2, 2);
      kb.addVariable(velocidad);
      velocidad.setType("output");
      velocidad.setDefaultValue(...); // de tipo float
      velocidad.setAccumulation("MAX");
      velocidad.setDefuzzifierName("COG");
      // Añadir otros términos lingüísticos ...

      // Reglas
      MamdaniRuleBaseType rb = new MamdaniRuleBaseType("Reglas");
      fis.addRuleBase(rb);

      // Regla 1
      FuzzyRuleType r1 = new FuzzyRuleType("Regla 1", "and", "MIN", 1.0f);
      AntecedentType ant1 = new AntecedentType();
      ant1.addClause(new ClauseType(angulo, angNG));
      // ...
      ant1.addClause(new ClauseType(velAngular, velAngZ));
      ConsequentType con1 = new ConsequentType();
      con1.addThenClause(velocidad, velPlaNG);
      r1.setAntecedent(ant1);
      r1.setConsequent(con1);
      rb.addRule(r1);

      // Añadir otras reglas ...

      // Inferencia
      angulo.setValue(...); // de tipo float
      velAngular.setValue(...); // de tipo float
      fis.evaluate();
      System.out.println(velocidad.getValue());
   }
}
```