Práctica de Lógica

1.- Introducción

El prototipo ***aima-java*** utilizado ya en prácticas anteriores, dispone de un módulo para la representación de conocimiento mediante lógica de primer orden o lógica de predicados. Este módulo se encuentra en aima-core.logic.fol, dónde FOL es la abreviatura de *First Order Logic*.

Durante esta clase práctica veremos los elementos más importantes de este módulo, así como cómo utilizarlo para representar conocimiento.

2.- Definición del dominio

Para poder utilizar el módulo de lógica de aima-java, es necesario especificar primero cuál es el dominio de nuestro problema. Es decir, debemos especificar las constantes, predicados y funciones que van a aparecer en nuestro modelo. Para ello, debemos añadir un nuevo método estático a la clase *DomainFactory*:

```
public static FOLDomain miDominio() {
            ……

}
```

El método empezará siempre creando un objeto de tipo *FOLDomain*, que será el valor devuelto al final.

```
public static FOLDomain miDominio() {
      FOLDomain dominio = new FOLDomain();
            ……
      return dominio;
}
```

Es en el cuerpo del método dónde se define el propio dominio. Para ello, la clase *domain* dispone de los métodos *addConstant*, *addFunction* y *addPredicate* de la propia clase *FOLDomain*:

```
public static FOLDomain miDominio () {
      FOLDomain dominio = new FOLDomain();
      dominio.addConstant("Constante1");
      dominio.addConstant("Constante2");

      dominio.addPredicate("p");
      dominio.addPredicate("q");
```

```
        dominio.addFunction("f");

        return dominio;
    }
```

Nótese que, a la hora de definir las funciones y predicados, no se especifica el número de argumentos que utiliza cada uno. Este hace que tengamos más libertad a la hora de escribir nuestras fórmulas, pero debemos tener cuidado de que un mismo predicado o una misma función reciba siempre el mismo número de parámetros.

Ejercicio:

Disponemos del siguiente conocimiento que queremos modelar mediante lógica de predicados:

1. *Todas aquellas personas que no son filósofos/as, son mortales.*
2. *Hay algunos filósofos que son mortales también.*
3. *Aunque hay otros que no lo son*
4. *Sócrates es una persona no mortal*

a) Formaliza los enunciados dados en términos de lógica de predicados:

$$\mathbf{1}.\ \forall x\ ((Persona(x) \wedge \neg Filosofo(x)) \rightarrow Mortal(x))$$

$$\mathbf{2}.\ \exists x\ (Filosofo(x) \wedge Mortal(x))$$

$$\mathbf{3}.\ \exists x\ (Filosofo(x) \wedge \neg Mortal(x))$$

$$\mathbf{4}.\ Persona(Socrates) \wedge \neg Mortal(Socrates)$$

b) Identifica las constantes, funciones y predicados utilizados y define el dominio en aima-java.

3.- Introducción de fórmulas en aima-java

La introducción de fórmulas en aima-java se hace mediante cadenas de texto donde cada conector se expresa como:

- Para todo x (∀x): FORALL x (…)
- Existe x (∃x): EXISTS x (…)
- Negación (¬): NOT …
- Y (∧): … AND …
- O (∨): … OR …
- Implica (→): … => …
- Doble implicación (↔): … <=> …
- (…)

Ejemplo:

```
" FORALL x (FORALL y (Animal(y) =>Loves(x,y)) => EXISTS y Loves(y,x))"”
```

4.- Unificación

Una de las funcionalidades que ofrece aima-java es la de calcular el unificador más general (u.m.g.) a partir de un conjunto de fórmulas ya dadas. Para ello, utiliza la clase *Unifier*.

Ejercicio:

a) Sigue los siguientes pasos:
   - Abre el fichero *FolDemo* situada en aima-gui.demo.logic
   - En el método main, comenta todas las líneas excepto “unifierDemo()”
   - Echa un vistazo al método *unifierDemo*().
   - Ejecútalo para comprobar que sale.

b) Cambia los enunciados por los siguientes y ejecuta de nuevo:

   Knows(Mother(Jane), x)
   Knows(y, Mother(Mother(John)))
   Knows(x, Mother(y))

   ¿Da el resultado esperado?

c) Cambia los enunciados por los siguientes y ejecuta de nuevo:

   Knows(Mother(Jane), x)
   Knows(y, Mother(Mother(John)))
   Knows(y, Mother(y))

   ¿Da el resultado esperado?

5.- Paso a Forma Normal de Skolem

En el propio fichero *FolDemo* se puede ver otra de las funcionalidades de aima-java, en este caso el paso a Forma Normal de Skolem.

Ejercicio:

a) Sigue los siguientes pasos:
- Abre el fichero *FolDemo* situada en aima-gui.demo.logic
- En el método main, comenta todas las líneas excepto "fOL_CNFConversion()"
- Echa un vistazo al método fOL_*CNFConversion*().
- Ejecútalo para comprobar que sale.

b) Modifica la función para que convierta a FNS el segundo enunciado del ejercicio descrito en el apartado 2. ¿Qué observas?

6.- Resolución General

Finalmente, la funcionalidad más interesante para la parte de representación del conocimiento es aquella que nos permite establecer una base de conocimiento y resolver distintas preguntas sobre esta base. Para ello, aima-java incorpora un método ("Two-Finger Method") para aplicar Resolución General mediante la estrategia de Green.

Para poder aplicarlo, además de definir el dominio como ya hemos hecho antes, es necesario definir la base de conocimiento. En el caso de lógica de predicados, especificar los enunciados que definen el conocimiento de que disponemos.

Para añadir una nueva base de conocimiento a aima-java, debemos crear un nuevo método estático en la clase *FOLKnowledgeBaseFactory*:

```
public static FOLKnowledgeBase
createMiBaseConocimiento(InferenceProcedure infp) {
                    ……

        }
```

El método empezará siempre creando un objeto de tipo *FOLKnowledgeBase*, que será el valor devuelto al final. En el constructor, debe especificarse el **dominio** que vamos a utilizar para definir la base de conocimiento:

```
public static FOLKnowledgeBase
createMiBaseConocimiento(InferenceProcedure infp) {
        FOLKnowledgeBase kb = new
        FOLKnowledgeBase(DomainFactory.miDominio(), infp);

        kb.tell("FORALL x (Persona(x) AND NOT Filosofo(x) => … )");
                    ……

        return kb;
```

```
}
```

En el fichero *FolDemo* podéis ver ejemplos de cómo hacer una pregunta a una base de conocimiento. Para ello, echa un vistazo al método *filosofosDemo1* de dicho fichero. Como se puede apreciar en el código, se debe indicar la base de conocimiento a utilizar y a continuación especificar la pregunta (formalizada en fol) que se desea realizar a la base de datos.

Ejercicio:

- Abre el fichero *FolDemo* situada en aima-gui.demo.logic
- En el método *main*, comenta todas las líneas excepto "fOL_TFMResolutionDemo ()"
- En el método *fOL_TFMResolutionDemo*, comenta todas las líneas excepto filosofosDemo1(new FOLTFMResolution());
- En el método *filosofosDemo1*, ¿cuál es la pregunta que se está haciendo a la base de conocimiento?
- Ejecuta el proyecto y comprueba el resultado obtenido.
- ¿Se ha obtenido la cláusula vacía? ¿Por qué?

7.- Ejercicio para entregar

El siguiente ejercicio debe entregarse durante la hora de clase y contará para el cálculo de la nota de prácticas voluntarias.

Dada la siguiente base de conocimiento:

Algunos dibujos animados no son infantiles.

Los dibujos animados no infantiles son para adultos.

Los niños que ven dibujos animados para adultos no los entienden y se aburren.

Los Simpsons son dibujos animados para adultos.

Juan es un niño y ve los Simpsons.

Utilizar aima-java para contestar a la siguiente pregunta:

¿Existe algún niño que se aburra?

Para ello:

a) Formaliza los enunciados en lógica de predicados
b) Define el dominio del problema en la clase *DomainFactory*
c) Define la base de conocimiento en la clase *FOLKnowledgeBaseFactory*

d) Define una función en la clase *FOLDemo* que aplique resolución para responder la pregunta

Una vez finalizado, se deberá generar un fichero en formato pdf donde aparezcan las tres funciones definidas en este ejercicio, así como copia de la salida ofrecida por aima-java.