fuzzyDL API

Fernando Bobillo and Umberto Straccia

The purpose of this document is to give some brief clues on how to use fuzzyDL reasoner from a Java application. We will concentrate on the use of fuzzyDL as a reasoner for fuzzy $\mathcal{SHIF}(\mathcal{D})$, avoiding for the sake of clarity many more complex features such as defuzzification or optimization of linear expressions.

The structure of this document is the following. Section 1 includes a sample template code which can be adapted according to the user needs. Basically, what is missing is to add the axioms to the knowledge base (Section 2), and to define the query (Section 3). As we will see, these tasks involve three non-trivial steps: create degrees of truth (Section 4), create individuals (Section 5), and create concepts (Section 6).

1 Template code

The following template code will be completed with the content of the next sections. There are two ways of building a fuzzy knowledge base. The easiest one is to read it from a file (see the syntax in http://straccia.info/software/fuzzyDL/documents/syntax.pdf, and some examples in http://straccia.info/software/fuzzyDL/examples/examples.html), but it is also possible to create an instance of the KnowledgeBase class and to add the axioms using the syntax in Section 2.

```
import java.io.*;
import java.util.*;

import fuzzydl.*;
import fuzzydl.exception.*;
import fuzzydl.milp.*;
import fuzzydl.parser.*;
import fuzzydl.util.*;

public class ClassUsingFuzzyDL
{
  public static void main(String[] args) throws IOException, ParseException
  {
      // Load options for the reasoner, using file "CONFIG"
      ConfigReader.loadParameters("CONFIG", new String[0]);

      try {
            // Option 1. Create KnowledgeBase and queries from scratch
            KnowledgeBase kb = new KnowledgeBase();
            // Add axioms to the KnowledgeBase. Example: kb.addAssertion(...);
            // Define queries: Query q = new MinSatisfiableQuery(Concept c);

            // Option 2. Read knowledge base and queries from file "fileName.txt"
            Parser parser = new Parser(new FileInputStream("filename.txt"));
```

```
            parser.Start();
            KnowledgeBase kb = parser.getKB();

            // The three latter lines can be replaced by the following one
            KnowledgeBase kb = Parser.getKB("fileName.txt")

            // Queries were also part of the file "fileName.txt"
            ArrayList <Query> queries = parser.getQueries();

            // After having created KB and queries, start logical inference
            kb.solveKB();

            // Solve a query q
            Solution result = q.solve(kb);

            // Print the result
            if (result.isConsistentKB())
                    System.out.println(q.toString() + result.getSolution());
            else
                    System.out.println("KB is inconsistent");

            // Optionally, show the language of the KB
            System.out.println("Language: " + kb.getLanguage());
      }
      catch (FuzzyOntologyException e)
            { System.out.println(e); }
  }
}
```

2 Adding axioms to the fuzzy KB

`fuzzyDL` supports the following fuzzy $\mathcal{SHIF}(\mathcal{D})$ axioms[1]:

```
// Concept assertion
kb.addAssertion(new Assertion(Individual a, Concept c, Degree d) );

// Role assertion
kb.addRelation(Individual a, String roleName, Individual b, Degree d);

// Definition of a concept, C = D
kb.defineConcept(String conceptName, Concept conc);

// Definition of a primitive concept, A = C
kb.defineAtomicConcept(String conceptName, Concept conc);

// GCI (default implication)
```

[1] For more information about individual and concept creation, see Section 5 and Section 6, respectively.

```
kb.implies(Concept conc1, Concept conc2, Degree deg);

// GCI (Goedel implication)
kb.g_implies(Concept conc1, Concept conc2, Degree deg);

// GCI (Lukasiewicz implication)
kb.l_implies(Concept conc1, Concept conc2, Degree deg);

// GCI (Kleene-Dienes implication)
kb.z_implies(Concept conc1, Concept conc2, Degree deg);

// Disjoint concepts
// temporaryConceptList is a vector of concepts
kb.addConceptsDisjoint(Vector temporaryConceptList);

// Range axiom
kb.roleRange(String roleName, Concept conc);

// Domain axiom
kb.roleDomain(String roleName, Concept conc);

// Functional roles
kb.roleIsFunctional(String roleName);

// Inverse roles
kb.addInverseRoles(String roleName, String invRoleName);

// Role hierarchies
kb.roleImplies(String roleChild, String roleParent, Degree deg);

// Reflexive roles
kb.roleIsReflexive(String roleName);

// Symmetric roles
kb.roleIsSymmetric(String roleName);

// Transitive roles
kb.roleIsTransitive(String roleName);
```

3 Creating queries

The query is the reasoning task that we want `fuzzyDL` to compute. For example, concept satisfiability, concept subsumption ... An example of query definition is the following:

```
Query query = MinSatisfiableQuery(Concept c);
```

Currently, `fuzzyDL` supports the following query constructors[2] (see http://straccia.info/software/fuzzyDL/documents/syntax.pdf for more details):

[2] There are also five additional queries to optimize expressions and to compute defuzzifications which are not considered here.

```
// Greatest lower bound of a concept assertion
new MinInstanceQuery(Concept c, Individual a);
new MaxInstanceQuery(Concept c, Individual a);

// Greatest lower bound of a role assertion
new MinRelatedQuery(Individual a, Individual b, String roleName);
new MaxRelatedQuery(Individual a, Individual b, String roleName);

// Concept subsumption
new MinSubsumesQuery(Concept c1, Concept c2, MinSubsumesQuery.LUKASIEWICZ);
new MinSubsumesQuery(Concept c1, Concept c2, MinSubsumesQuery.GODEL);
new MinSubsumesQuery(Concept c1, Concept c2, MinSubsumesQuery.ZADEH);
new MaxSubsumesQuery(Concept c1, Concept c2, MaxSubsumesQuery.LUKASIEWICZ);
new MaxSubsumesQuery(Concept c1, Concept c2, MaxSubsumesQuery.GODEL);
new MaxSubsumesQuery(Concept c1, Concept c2, MaxSubsumesQuery.ZADEH);

// Concept satisfiability
new MinSatisfiableQuery(Concept c);
new MinSatisfiableQuery(Concept c, Individual a);
new MaxSatisfiableQuery(Concept c);
new MaxSatisfiableQuery(Concept c, Individual a);
```

4 Creating degrees of truth

In `fuzzyDL`, the degree of a fuzzy axiom can be *(i)* a rational number in $[0, 1]$, *(ii)* a variable, *(iii)* an already defined truth constant, and *(iv)* a linear expression.

```
// 1. Rational number
Degree deg = Degree.getDegree(Double num);

// 2. Variable
Degree deg = Degree.getDegree(kb.milp.getVariable(varName));

// 3. Truth constant, currently it is not allowed in the API

// 4. Linear expression
Degree deg = Degree.getDegree(Expression expr);
```

How to define an expression is not covered here, see `Expression` class for details.

5 Creating individuals

Use the the following method in `KnowledgeBase` class.

```
Individual getIndividual(String indName);
```

An example of use:

```
Individual a = kb.getIndividual("umberto");
```

6 Creating concepts

Now we describe how to build complex $\mathcal{SHIF}(\mathcal{D})$ concepts (the expressions below return a valid concept). `fuzzyDL` supports other constructors which are not being considered here. Among them, `Concept` class have static constructors to build at least value, at most value, exact value, weighted and threshold concepts; while the class `WeightedSumConcept` allows to build weighted sum concepts.

```
// Atomic concept
Concept c = new Concept(String name);

// Concept negation
Concept c = Concept.complement(Concept c);

// Concept conjunction (default, Goedel and Lukasiewicz)
Concept.and(Concept c1, Concept c2);
Concept.g_and(Concept c1, Concept c2);
Concept.l_and(Concept c1, Concept c2);

// Concept disjunction (default, Goedel and Lukasiewicz)
Concept.or(Concept c1, Concept c2);
Concept.g_or(Concept c1, Concept c2);
Concept.l_or(Concept c1, Concept c2);

// Concept implication (default, Goedel, Lukasiewicz and Kleene-Dienes)
Concept.implies(Concept c1, Concept c2);
Concept.g_implies(Concept c1, Concept c2);
Concept.l_implies(Concept c1, Concept c2);
Concept.z_implies(Concept c1, Concept c2);

// Existential restriction
Concept.some(String role, Concept c);

// Universal restriction
Concept.all(String role, Concept c);

// Rough concepts
Concept.upperApprox(String fuzzySimRelation, Concept c);
Concept.lowerApprox(String fuzzySimRelation, Concept c);

// Local reflexivity concept
Concept.self(String role);

// Modified concepts
Modifier m = new LinearModifier(String name, double b);
Modifier m = new TriangularModifier(String name, double a, double b, double c);
Concept d = m.modify(Concept c);
```

```
// Concrete fuzzy concepts (use only in existential and universal restrictions)
new CrispConcreteConcept(String conceptName, double k1, double k2,
                         double a, double b);
new LeftConcreteConcept(String conceptName, double k1, double k2,
                        double a, double b);
new RightConcreteConcept(String conceptName, double k1, double k2,
                         double a, double b);
new TriangularConcreteConcept(String conceptName, double k1, double k2,
                              double a, double b, double c);
new TrapezoidalConcreteConcept(String conceptName, double k1, double k2,
                               double a, double b, double c, double d);

// Fuzzy numbers (use only in existential and universal restrictions)
new TriangularFuzzyNumber(String conceptName, double a, double b, double c);

// The range of all fuzzy numbers has to be fixed
TriangularFuzzyNumber.setRange(double min, double max);
```

7 Show expressions

fuzzyDL can show the values in an optimal solution.

```
// Show the value of a variable
kb.milp.showVars.addVariable(Variable var, String varName);

// Show the value of all fillers of a concrete role
kb.milp.showVars.addConcreteFillerToShow(String roleName);

// Show the value of the concrete filler of some individual
kb.milp.showVars.addConcreteFillerToShow(String roleName, String indName);

// Show the membership to several atomic concepts of all
// the fillers of a concrete role
kb.milp.showVars.addAbstractFillerToShow(
        String roleName, HashSet<String> conceptNames);

// Show the membership to several atomic concepts of the
// fillers of a concrete role for some individual
kb.milp.showVars.addAbstractFillerToShow(String roleName,
        String indName, HashSet<String> conceptNames);

// Show all instances of an atomic concept
kb.milp.showVars.addConceptToShow(String conceptName);

// Show all atomic concepts of an individual
kb.milp.showVars.addIndividualToShow(String indName);
```