fuzzyDL API

Fernando Bobillo and Umberto Straccia

October 16, 2015

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 8), 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 8.

Note that it is not necessary to clone the KB to solve several queries over it, as the newer versions of fuzzyDL are able to do it automatically.

```
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 FuzzyOntologyException,
                                                  InconsistentOntologyException,
                                                  IOException, ParseException
   {
      // Load options for the reasoner, using file "CONFIG"
      ConfigReader.loadParameters("CONFIG", new String[0]);

      // 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")

        // Option 3. Clone an existing KB
        KnowledgeBase kb2 = kb.clone();

        // 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 ( (q instanceof AllInstancesQuery) &&
             kb.getIndividuals().values().isEmpty() )
           System.out.println(q.toString() + " No individuals in the KB");
        else {
           if (result.isConsistentKB()) {
              for (int i=0; i<q.getIndividuals().size(); i++)
                 System.out.println(q.getIndividuals().get(i) + " : " +
                                    q.getDegrees().get(i));
              System.out.println(/*q.toString() +*/ result.getSolution());
           }
           else
              System.out.println("KB is inconsistent");
        }

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

        // Optionally, show the time and language of the KB
        System.out.println("Time (s): " + query.getTotalTime());
        System.out.println("Language: " + kb.getLanguage());
    }
}
```

2 Adding axioms to the fuzzy KB

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

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

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

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

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

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

// Inverse functional roles
```

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

```
kb.roleIsInverseFunctional(String roleName);

// 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):

```
// KB satisfiability
new KbSatisfiableQuery();

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

// Greatest lower bound of a concept for every individual
// Use methods getIndividuals and getDegrees to loop over the solutions
new AllInstancesQuery(Concept c);

// 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);
```

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

```
// 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 (see `Expression` class for to define expressions).

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

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

// 3. Linear expression
Degree deg = Degree.getDegree(Expression expr);

// 4. Truth constant currently is not allowed in the API
```

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 fuzzy $\mathcal{SHIF}(\mathcal{D})$ concepts (the expressions below return a valid concept), together with some additional fuzzy and fuzzy rough concepts.

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

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

// Concept conjunction (default, Goedel and Lukasiewicz)
Concept.and(Concept c1, Concept c2);
Concept.gAnd(Concept c1, Concept c2);
Concept.lAnd(Concept c1, Concept c2);
```

```
// Concept disjunction (default, Goedel and Lukasiewicz)
Concept.or(Concept c1, Concept c2);
Concept.gOr(Concept c1, Concept c2);
Concept.lOr(Concept c1, Concept c2);

// Concept implication (default, Goedel, Lukasiewicz and Kleene-Dienes)
Concept.implies(Concept c1, Concept c2);
Concept.gImplies(Concept c1, Concept c2);
Concept.lImplies(Concept c1, Concept c2);
Concept.zImplies(Concept c1, Concept c2);

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

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

// Top and bottom concepts
Concept.CONCEPT_TOP
Concept.CONCEPT_BOTTOM

// 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);
kb.addModifier(String conceptName, Modifier mod);
Concept d = m.modify(Concept c);

// Weighted and threshold concepts
WeightedConcept d = Concept.weightedConcept(
    double weight, Concept concept);
Concept.posThreshold(double w, Concept c)
Concept.negThreshold(double w, Concept c)
Concept.extendedPosThreshold(Variable w, Concept c)
Concept.extendedNegThreshold(Variable w, Concept c)

// Aggregation operators
WeightedSumConcept d = new WeightedSumConcept(
    ArrayList<Double> weights, ArrayList<Concept> concepts);
WeightedSumZeroConcept d = new WeightedSumConcept(
    ArrayList<Double> weights, ArrayList<Concept> concepts);
OwaConcept d = new OwaConcept(ArrayList<Double> weights,
    ArrayList<Concept> concepts);
QowaConcept d = new QowaConcept(FuzzyConcreteConcept f,
    ArrayList<Concept> concepts);
ChoquetIntegral d = new ChoquetIntegral(ArrayList<Double> weights,
```

```
    ArrayList<Concept> concepts);
SugenoIntegral d = new SugenoIntegral(ArrayList<Double> weights,
    ArrayList<Concept> concepts);
QsugenoIntegral d = new QsugenoIntegral(ArrayList<Double> weights,
    ArrayList<Concept> concepts);

// Fuzzy rough concepts
Concept.lowerApprox(String fuzzySimRelation, Concept c);
Concept.looseLowerApprox(String fuzzySimRelation, Concept c);
Concept.tightLowerApprox(String fuzzySimRelation, Concept c);
Concept.upperApprox(String fuzzySimRelation, Concept c);
Concept.looseUpperApprox(String fuzzySimRelation, Concept c);
Concept.tightUpperApprox(String fuzzySimRelation, Concept c);

// Concrete fuzzy concrete concepts
// (Use them 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);
new LinearConcreteConcept(String conceptName, double k1, double k2,
                          double c);
new ModifiedConcreteConcept(String conceptName, LinearModifier mod,
                            FuzzyConcreteConcept fMod);

// If there fuzzy concrete concepts, we must assert it
kb.concreteFuzzyConcepts = true;

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

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

// Datatype restrictions, where object o can be:
// * Boolean b,
// * Double d,
// * Integer i,
// * Variable v,
// * FeatureFunction fun,
// * TriangularFuzzyNumber tri
kb.addDatatypeRestriction(Concept.AT_MOST_VALUE,
    FeatureFunction fun, String role);
```

```
kb.addDatatypeRestriction(Concept.AT_LEAST_VALUE,
    FeatureFunction fun, String role);
kb.addDatatypeRestriction(Concept.EXACT_VALUE,
    FeatureFunction fun, String role);
```

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 the concepts of all the fillers
// of an abstract role
kb.milp.showVars.addAbstractFillerToShow(String roleName);

// Show the membership to the concepts of all the fillers
// of an abstract role for some individual
kb.milp.showVars.addAbstractFillerToShow(String roleName, String indName);

// 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);
```

8 Saving fuzzy KBs

Fuzzy KBs can be translated into a file with a fuzzy ontology in fuzzyDL syntax

```
kb.saveToFile("outputFile.txt")
```

fuzzyDL API can translate fuzzy ontologies in fuzzyDL syntax into fuzzy OWL 2, i.e., OWL 2 with annotations encoding fuzzy information.

```
String inputFile = "fuzzydl.txt";
String outputFile = "fuzzyOntology.owl";
FuzzydlToOwl2 f = new FuzzydlToOwl2(inputFile, outputFile);
f.run();
```