

# Web Interface for RFuzzy Query-Answering

Lu JingWei

February 21, 2011

## 1 Introduction

Logic studies the notions of consequence. It expresses natural language in different formal logic, such as propositional logic, predicate logics, modal propositional/predicate logic, many-valued propositional/predicate logic. Logic deals with set of formulas and the relation of consequence among them. The task of formal logic is to represent all this by means of well-defined logical calculi admitting exact investigation. The intuitive distinction among different formal logics is their expressivity of natural language. In real world, most information is vague, most of which can not be represented by classical logic. For instance, we represent sentence “The patient is young.” by several well-known classical logic. Propositional Logic can represent it as proposition  $p$ . Semantically, it only can express true or false to the whole statement, but not the truth value of “young”. First Order Logic is able to express it as *young(the patient)*, but only can assign true or false to vague information “young”. However, Fuzzy Logic is an answer to represent imprecise information. The sentence “The patient is young.” is true to some degree in unit interval  $[0, 1]$  - the lower the age of the patient, the more the sentence is true. Truth of Fuzzy Logic is a matter of degree that the element belongs to a fuzzy concept. The degree is in  $[0, 1]$ , where 0 means absolute “False”, 1 means absolute “True”. Thus, the truth degrees is coded in real number between 0 and 1.

Logic programming [Llo87] has been successfully used in knowledge representation and reasoning for decades. Indeed, world data is not always perceived in a crisp way. Information that we gather might be imperfect, uncertain, or fuzzy in some other way. Hence the management of uncertainty and fuzziness is very important in knowledge representation.

Introducing Fuzzy Logic into Logic Programming has provided the development of several fuzzy systems over Prolog. These systems replace its inference mechanism, SLD-resolution, with a fuzzy variant that is able to handle partial truth. Most of these systems implement the fuzzy resolution introduced by Lee in [Lee72], as the Prolog-Elf system, the FRIL Prolog system [IK85] and the F-Prolog language [LL90]. However, there is no common method for fuzzifying Prolog, as noted in [ZM89].

One of the most promising fuzzy tools for Prolog is RFuzzy Framework. The most important advantages against the other approaches are:

1. A truth value is represented as a real number in unit interval  $[0, 1]$ , satisfying certain constraints.
2. A truth value is propagated through the rules by means of an aggregation operator. The definition of this aggregation operator is general and is subsumes conjunctive operators (triangular norms [KP00] like min, prod, etc.), disjunctive operators [ETC95] (triangular co-norms, like max, sum, etc.), average operators (average as arithmetic average, quasi-linear average, etc.) and hybrid operators (combinations of the above operators [APC02]).
3. Crisp and fuzzy reasoning are consistently combined
4. It provides some interesting improvements with respect to FLOPER [JM08, Mor06]: default values, partial default values, typed predicates and useful syntactic sugar (for presenting facts, rules and functions).

Fuzzy Query-Answering System (FQAS) is built on RFuzzy Framework, which inherits the advantages of RFuzzy Framework. Besides, *quantification* expression is added into FQAS. In order to query under RFuzzy Framework, we have to write a ciao prolog program, which requires the high level knowledge from users. FQAS has an user-friendly interface to overcome this disadvantage.

In Fuzzy Query-Answering System, it offers user to load crisp database, and to do configurations over crisp database, such as defining negations, quantifications, and fuzzy concepts. The simple or complex fuzzy queries can be generated by choosing negations, quantifications and fuzzy concepts which are defined during configuration. According to the functionality of FQAS, it is built into three modules, which are Loading File Module, Configuration Module and Query-Answering Module.

In this report, we first present RFuzzy Framework in section 2. In section 3, Technical details of this project are shown, where three main modules Loading File Module, Configuration Module and Query-Answering Module are presented in section 3.1, 3.2 and 3.3, respectively. The Conclusions of Fuzzy Query-Answering System are given in the last section.

## 2 Background

RFuzzy framework is a Prolog-based tool for representing and reasoning with fuzzy information. We introduce the RFuzzy framework presented by Susana Muñoz-Hernández [MHPCS10] in this chapter to familiarize the reader with the syntax and semantics which we work on in Fuzzy Query-Answering project.

### 2.1 Syntax

From  $\Sigma$  a set of function symbols and a set of variables  $V$ , the *term universe*  $TU_{\Sigma,V}$  is built, whose elements are *terms*. It is the minimal set such that each variable is a term and terms are closed under  $\Sigma$  operations. In particular, constant symbols are terms, which is a function with arity 0. The creation procedure of *term universe* is described in definition 2.3.1.

*Term base*  $TB_{\Pi,\Sigma,V}$  is defined from a set of predicate symbols and *term universe*  $TU_{\Sigma,V}$ . The elements in the *term base* are called *atoms*, which are predicates whose arguments are elements of  $TU_{\Sigma,V}$ . The building procedure is described in definition 2.3.2.

Atoms and terms are called *ground* if they do not contain variables. The *Herbrand universe*  $\mathbb{HU}$  is the set of all ground terms, and the *Herbrand base*  $\mathbb{HB}$  is the set of all atoms with arguments from the *Herbrand universe*.

$\Omega$  is a set of *many-valued connectives* and includes,

1. conjunctions  $\&_1, \&_2, \dots, \&_k$
2. disjunctions  $\vee_1, \vee_2, \dots, \vee_l$
3. implications  $\leftarrow_1, \leftarrow_2, \dots, \leftarrow_m$
4. aggregations  $@_1, @_2, \dots, @_n$
5. real numbers  $v \in [0, 1] \subset \mathbb{R}$ . These connectives are of arity 0, that is,  $v \in \Omega^{(0)}$ .

While  $\Omega$  denotes the set of connectives symbols,  $\hat{\Omega}$  denotes a set of associated truth functions. Instances of connective symbols and their truth functions are denoted by  $F$  and  $\hat{F}$  respectively, so  $F \in \Omega$ , and  $\hat{F} \in \hat{\Omega}$ .

**Definition 2.1. (fuzzy clauses).** A fuzzy clause is written as,

$$A \stackrel{c, F_c}{\leftarrow} F(B_1, \dots, B_n)$$

where  $A \in TB_{\Pi,\Sigma,V}$  is called *head*, and  $B_i \in TB_{\Pi,\Sigma,V}$  is called *body*,  $i \in [1, n]$ .  $c \in [0, 1]$  is *credibility value*, and  $F_c \in \{\&_1, \dots, \&_k\} \subset \Omega^{(2)}$  and  $F \in \Omega^{(n)}$  are *connective symbols for the credibility value and the body, respectively*.

A *fuzzy fact* is a special case of a clause where  $c = 1$ ,  $F_c$  is the usual multiplication of real numbers “.” and  $n = 0$ . It is written as  $A \leftarrow v$ , where the  $c$  and  $F_c$  are omitted.

**Example 2.2.**

$$\text{good} - \text{destination}(X) \stackrel{1.0}{\leftarrow} .(\text{nice} - \text{weather}(X), \text{many} - \text{sight}(X)).$$

This fuzzy clause models to what extent cities can be deemed good destinations - the quality of the destination depends on the weather and the availability of sights. The credibility value of the rule is 1.0, which means that we have no doubt about this relationship. The connectives used here in both cases is the usual multiplication of real numbers.

**Definition 2.3. (default value declaration).** A default value declaration for a predicate  $p \in \Pi^{(n)}$  is written as

$$\text{default}(p/n) = [\delta_1 \text{ if } \varphi_1, \dots, \delta_m \text{ if } \varphi_m]$$

where  $\delta_i \in [0, 1]$  for all  $i$ . The  $\varphi_i$  are FOL (First-Order Logic) formulas restricted to terms from  $TU_{\Sigma, V_p}$ , the predicates  $=$  and  $\neq$ , the symbol true and the junctors  $\vee$  and  $\wedge$  in their usual meaning.

**Example 2.4.**

$$\text{default}(\text{nice} - \text{weather}(X)) = 0.5$$

**Definition 2.5. (type declaration).** Types are built from terms  $t \in \mathbb{H}\mathbb{U}$ . A term type declaration assigns a type  $\tau \in \mathcal{T}$  to a term  $t \in \mathbb{H}\mathbb{U}$  and is written as  $t : \tau$ . A predicate type declaration assigns a type  $(\tau_1, \dots, \tau_n) \in \mathcal{T}^n$  to a predicate  $p \in \Pi^n$  and is written as  $p : (\tau_1, \dots, \tau_n)$ , where  $\tau_i$  is the type of  $p$ 's  $i$ -th argument. The set composed by all term type declarations and predicate type declarations is denoted by  $T$ .

**Example 2.6.** Suppose that the set of types is  $\mathcal{T} = \{\text{City}, \text{Continent}\}$ , here are some examples of term type declaration and predicate type declaration.

$$\text{madrid} : \text{City}, \text{sydney} : \text{City}$$

$$\text{africa} : \text{Continent}, \text{america} : \text{Continent}$$

$$\text{nice} - \text{weather} : (\text{City})$$

$$\text{city} - \text{continent} : (\text{City}, \text{Continent})$$

In this example, “madrid”, “sydney”, “africa” and “america” are ground terms. “madrid” and “sydney” have type *City*. “africa” and “america” have type *Continent*. “nice-weather” and “city-continent” are fuzzy predicates. “nice-weather” is of type *(City)* and “city-continent” is of type *(City, Continent)*.

**Definition 2.7. (well-typed).** A ground atom  $A = p(t_1, \dots, t_n) \in \mathbb{H}\mathbb{B}$  is well\_typed with respect to  $T$  **iff**  $p : (\tau_1, \dots, \tau_n) \in T$  implies that the term type declaration  $(t_i : \tau_i) \in T$  for  $1 \leq i \leq n$ .

A ground clause  $A \stackrel{c}{\leftarrow} F(B_1, \dots, B_n)$  is well\_typed w.r.t  $T$  **iff** all  $B_i$  are well\_typed for  $1 \leq i \leq m$  implies that  $A$  is well\_typed. A non-ground clause is well\_typed **iff** all its ground instances are well\_typed.

**Definition 2.8. (well-defined RFuzzy program).** A fuzzy program  $P = (R, D, T)$  is called well-defined **iff**

- for each predicate symbol  $p/n$ , there exist both a predicate type declaration and a default value declaration.
- all clauses in  $R$  are well-typed.
- for each default value declaration

$$\text{default}(p(X_1, \dots, X_n)) = [\delta_1 \text{ if } \varphi_1, \dots, \delta_m \text{ if } \varphi_m].$$

the formulas  $\varphi_i$  are pairwise contradictory and  $\varphi_1 \vee \dots \vee \varphi_m$  is a tautology.

## 2.2 Semantics

A valuation  $\sigma : V \rightarrow \mathbb{HB}$  is an assignment of ground terms to variables. Each valuation  $\sigma$  uniquely constitutes a mapping  $\hat{\sigma} : TB_{\Pi, \Sigma, V} \rightarrow \mathbb{HB}$  that is defined in the obvious way.

A fuzzy Herbrand interpretation of a fuzzy logic program is a mapping  $I : \mathbb{HB} \rightarrow \mathbb{T}$  that assigns truth values to ground atoms.

**Definition 2.9. (Model).** Let  $P = (R, D, T)$  be a fuzzy logic program.

For a clause  $r \in R$  of the form  $A \xrightarrow{c, F_c} F(B_1, \dots, B_n)$ , then  $I$  is a model of the clause  $r$  and is written as

$$I \models A \xrightarrow{c, F_c} F(B_1, \dots, B_n)$$

**iff** for all valuations  $\sigma$ :

$$\text{if } I(\sigma(B_i)) = v_i > 0 \text{ for all } i \text{ then } I(\sigma(A)) \geq v'$$

where  $v' = \hat{F}_c(c, \hat{F}(v_1, \dots, v_n))$ .

For a clause  $r \in R$  of the form  $A \leftarrow v$ , then  $I$  is a model of a clause  $r$  and is written as,

$$I \models A \leftarrow v$$

**iff** for all valuations  $\sigma$ :

$$I(\sigma(A)) \geq v$$

For a default value declaration  $d \in D$ , then  $I$  is a model of the default value declaration  $d$  and is written as,

$$I \models \text{default}(p/n) = [\delta_1 \text{ if } \varphi_1, \dots, \delta_m \text{ if } \varphi_m]$$

**iff** for all valuation  $\sigma$  :

$$I(\sigma(p(X_1, \dots, X_n))) \geq \delta_i$$

where  $\varphi_j$  is the only formula that holds for  $\sigma(\varphi_j)(1 \leq j \leq m)$ .

$I \models R$  **iff**  $I \models r$  for all  $r \in R$  and similarly,  $I \models D$  **iff**  $I \models d$  for all  $d \in D$ . Finally,  $I \models P$  **iff**  $I \models R$  and  $I \models D$ .

### 3 Dive into Fuzzy Query-Answering Web Interface

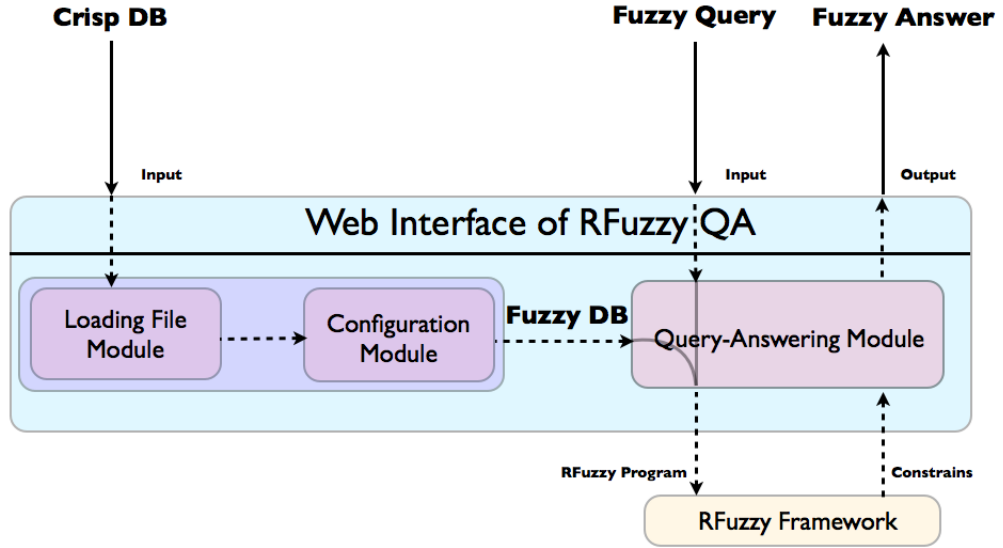


Figure 1: Fuzzy Query-Answering System

The framework of Fuzzy Query-Answering System (FQAS) is given in figure 1. FQAS receive crisp database and fuzzy query as input, and output fuzzy answer, which means users can ask fuzzy queries over crisp database. In FQAS, *Loading File Module* (LFM) accepts crisp database, which parse the schema of database; *Configuration Module* (CM) offers users to create fuzzy concepts over crisp database, as well as negation and quantification. Simply speaking, it generates a virtual fuzzy database upon crisp database; *Query-Answering Module* (QAM) takes virtual fuzzy database and fuzzy query as input and generates a RFuzzy prolog program, which is passed to RFuzzy Framework to do the reasoning, QAM parses the result from RFuzzy Framework as fuzzy answer of FQAS.

In this section, the main three modules are presented in detail in section 3.1,

3.2 and 3.3, respectively.

### 3.1 Loading File Module

Loading File Module offers users to choose a crisp database and its schema from local file system. It parses database schema and generates a table called Attribute to store information (name of attribute, type of attribute, range of attribute), on which fuzzy concepts are built.

#### 3.1.1 Database and its schema

Crisp database is in a prolog file with extension “.pl”, and its schema is in a XML file with extension “.xml”. Crisp database is simply one predicate prolog program, the predicate is the name of table, and its arguments are terms of atom. Each atom is considered as an entry in the table. The schema of database is presented in .xml file, which stores table name, primary key and attributes information. There are examples of crisp database in .pl file and its schema in .xml file.

**Example 3.1.** *In .pl file, house is the only predicate, and it is the name of table. All the atoms in .pl file are entries of the table house.*

```
house(lfs2168, 'apartment', 114, 5, 630000, 2, 5700).
house(lfs2144, 'apartment', 77, 3, 420000, 7, 3500).
house(lfs2147, 'apartment', 80, 2, 675000, 12, 200).
house(lfs2145, 'apartment', 224, 8, 790000, 20, 100).
house(c358, 'apartment', 74, 3, 340000, 5, 3100).
house(lfs2110, 'apartment', 415, 9, 2500000, 8, 2400).
house(lfs2124, 'apartment', 63, 2, 275000, 15, 450).
house(lfs2123, 'apartment', 62, 3, 285000, 6, 1000).
house(lfs2111, 'villa', 700, 10, 1100000, 9, 4500).
house(lfs2047, 'villa', 1750, 11, 1650000, 15, 1000).
house(lfs2041, 'villa', 4000, 13, 2500000, 4, 1800).
house(es13462, 'villa', 600, 6, 4000000, 6, 1500).
house(lfs1942, 'villa', 900, 10, 3100000, 3, 3400).
house(lfs1917, 'villa', 210, 5, 590000, 13, 5000).
house(lfb143, 'villa', 1200, 9, 2750000, 7, 4000).
house(5607/152, 'town_house', 161, 7, 815000, 6, 1200).
house(es13340, 'town_house', 1025, 8, 2800000, 25, 7000).
house(lfs1939, 'town_house', 860, 9, 1800000, 14, 2400).
house(lfs1938, 'town_house', 520, 11, 1990000, 19, 80).
house(lfs2155, 'villa', 2300, 9, 3000000, 13, 800).
```

**Example 3.2.** *As shown in .xml file, the name of table is house, the primary key is house\_code. The attributes of table house are type, size, number of rooms, price, distance from center, close to beach. The information about domain of each attribute is also included.*

```

<?xml version="1.0"?>
<House>
  <PrimaryKey>house_code</PrimaryKey>
  <AttrDom>
    <Attribute>type</Attribute>
    <Domain>
      <Type>D</Type>
      <Range>["apartment","villa","town_house"]</Range>
    </Domain>
  </AttrDom>
  <AttrDom>
    <Attribute>size</Attribute>
    <Domain>
      <Type>C</Type>
      <Range>[(0,2500)]</Range>
    </Domain>
  </AttrDom>
  <AttrDom>
    <Attribute>number_of_rooms</Attribute>
    <Domain>
      <Type>D</Type>
      <Range>[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]</Range>
    </Domain>
  </AttrDom>
  <AttrDom>
    <Attribute>price</Attribute>
    <Domain>
      <Type>C</Type>
      <Range>[(0,5000000)]</Range>
    </Domain>
  </AttrDom>
  <AttrDom>
    <Attribute>distance_from_center</Attribute>
    <Domain>
      <Type>C</Type>
      <Range>[(0,30)]</Range>
    </Domain>
  </AttrDom>
  <AttrDom>
    <Attribute>close_to_beach</Attribute>
    <Domain>
      <Type>C</Type>
      <Range>[(0,2000)]</Range>
    </Domain>
  </AttrDom>
</House>

```



### 3.1.2 How to use interface of Loading File Module

The web interface of Loading File Module is shown in figure 2. Three steps to load a database.

1. Choose crisp database from Db selection box
2. Choose corresponding database schema from Xml selection box
3. Click Button Load

In the last step, system stores file information in model “InputFile” and parses .xml file into two models, one is “Table”, the other is “Attribute”. All the models are in file models.py.

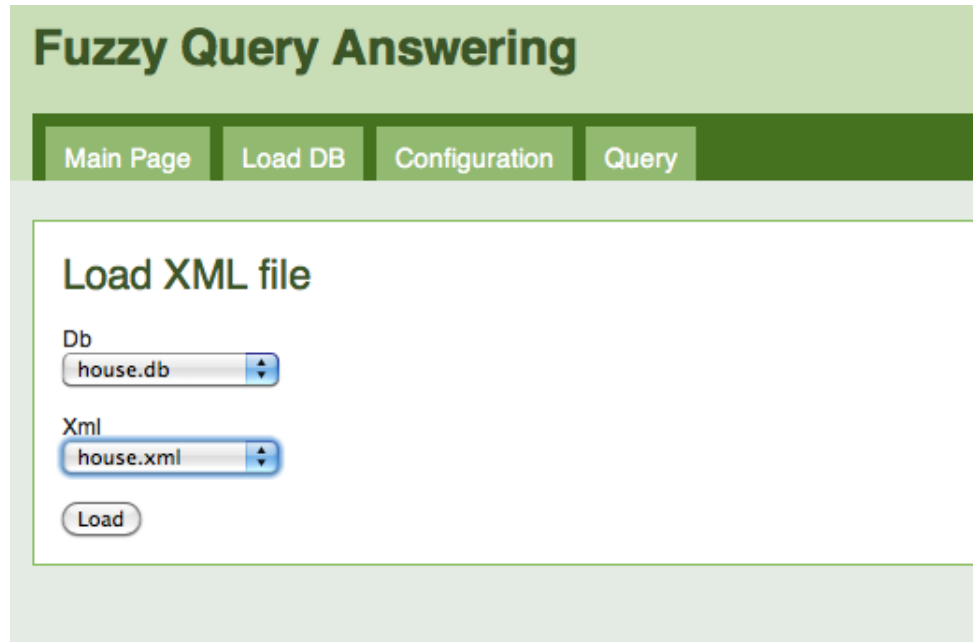
The image shows a web application titled "Fuzzy Query Answering". It has a navigation bar with four tabs: "Main Page", "Load DB", "Configuration", and "Query". The "Load DB" tab is currently selected. Below the navigation bar, there is a section titled "Load XML file". This section contains two dropdown menus. The first is labeled "Db" and has "house.db" selected. The second is labeled "Xml" and has "house.xml" selected. Below these dropdowns is a button labeled "Load".

Figure 2: Loading File Interface

## 3.2 Configuration Module

Configuration Module functions the configuration of fuzzy concepts over crisp database, negation and quantification of natural language. The crisp database is the one load it in the Loading File Module. Users are entitled to create and edit fuzzy concepts such as “beautiful”, “old” by defining fuzzy functions over attributes in crisp database. Negations and quantifications can be created by defining the their functions over unit interval  $[0, 1]$ .

### 3.2.1 Fuzzy Concept

Fuzzy concept is created over one of attributes in crisp database. For example, we can create fuzzy concept “expensive” over attribute “price”. In this section, we formalize definition of fuzzy concept.

**Definition 3.3. Fuzzy Concept** Let  $\mathcal{A}$  be a set of attributes, and  $\mathcal{D}$  be a set of domain, that is  $\mathcal{D} = \{(type, range) \mid type = \{Continuous, Discrete\}\}$ .  $g_{AD}$  is a function mapping from attributes to domains, that is,  $\mathcal{A} \rightarrow \mathcal{D}$ . Suppose there exists fuzzy function  $f_D$  is  $\mathcal{D} \rightarrow [0, 1]$ .

A fuzzy concept  $FC$  is defined as a function  $f_{FC} : \mathcal{A} \rightarrow [0, 1]$ . Indeed,  $f_{FC}(att) = f_D(g_{AD}(att))$ , where  $att \in \mathcal{A}$ .

**Example 3.4. Create fuzzy concept “big”** As shown in *house.xml*, there exists an attribute “size” with its type “Continuous (C)” and its range “[0,2500]”, which means an interval from 0 to 2500.

```
<AttrDom>
  <Attribute>size </Attribute>
  <Domain>
    <Type>C</Type>
    <Range>[(0,2500)]</Range>
  </Domain>
</AttrDom>
```

A fuzzy concept “big” can be created over attribute “size” by defining fuzzy functions from domain  $[(0,2500)]$  to  $[0,1]$ . For example,

$$big(x) = \begin{cases} 0.002000 \times x & x \in (0.000000, 50.000000] \\ 0.003333 \times x - 0.066667 & x \in (50.000000, 80.000000] \\ 0.002500 \times x & x \in (80.000000, 120.000000] \\ 0.001250 \times x + 0.150000 & x \in (120.000000, 200.000000] \\ 0.001000 \times x + 0.200000 & x \in (200.000000, 500.000000] \\ 0.000200 \times x + 0.600000 & x \in (500.000000, 1500.000000] \\ 0.000100 \times x + 0.750000 & x \in (1500.000000, 2500.000000] \end{cases}$$

### 3.2.2 Fuzzy Quantification

Fuzzy quantifier can be defined as second order fuzzy relations, which is formalized as  $D : \mathcal{F}(U)^n \rightarrow [0, 1]$ , where  $U$  is domain of interest, and  $\mathcal{F}(U)$  is a set of all fuzzy subsets of  $U$ . In natural language, most of the statements with quantifiers takes less than 3 or 4 fuzzy subsets as arguments, such as “very tall pretty girls”, “extremely strict professors”. We express the simplest statements taking 2 fuzzy subsets as parameters, then quantifier  $Q$  is defined as a function  $Q : \mathcal{F}(U) \times \mathcal{F}(U) \rightarrow [0, 1]$ . In most of such statements, the last concept is not really fuzzy one, but crisp. Normally, the value of quantifier function will not be affected by the crisp concept. The statement “All men are tall.” is expressed as

$$\text{all}(\text{men}, \text{tall}) = \inf\{\max(1 - \text{men}(e), \text{tall}(e)), e \in U\}$$

, where the value of **men** doesn't actually affect on the final result. There is a linguistics reason here. In computational linguistics, dependency among words are considered as foundation to apply linguistics function over those words. For example, in statement "very tall men", "very" and "men" are not dependent on each other, but "very" and "tall" have dependency, so do "tall" and "persons". Therefore, the quantifier function "very" is not defined dependent on crisp set "men" according to the linguistics meaning.

Thus, we redefine quantifier without considering the crisp concept. It is formalized as  $Q : \mathcal{F}(U) \rightarrow [0, 1]$ . Even though, this definition is still a second order fuzzy set. In order to implement quantification in RFuzzy Framework, we need first order fuzzy predicate.

**Definition 3.5. Fuzzy Quantification defined in First Order Fuzzy Predicate** Let  $U$  be a set of fuzzy concepts, which is simply a set of names of fuzzy concepts.  $f : U \rightarrow [0, 1]$  is function of fuzzy concept.  $\mathcal{F}(U)$  is the set of all functions of all fuzzy concepts.  $Q$  is a set of quantifications, which was defined as  $Q : \mathcal{F}(U) \rightarrow [0, 1]$ . At present, for an arbitrary quantification  $q \in Q$ , for each fuzzy concept  $c \in U$ , we create a new quantification  $q_c : [0, 1] \rightarrow [0, 1]$ , where the first  $[0, 1]$  is the range of function  $f_c$  of fuzzy concept  $c$ .

**Example 3.6. Create quantification "very" for fuzzy concept "big"** As shown in the example 3.4, "big" is a fuzzy concept and its function is  $f(\text{big})$ . Suppose very is a fuzzy quantifier. Then we create a new fuzzy quantifier  $\text{very}_{\text{big}}$ , which is a function  $[0, 1] \rightarrow [0, 1]$ .

$$\text{very}_{\text{big}}(x) = \begin{cases} 0 & \text{if } 0 \leq x < 0.4 \\ 7/8 * x & \text{if } 0.4 \leq x < 0.8 \\ x & \text{if } 0.8 \leq x \leq 1.0 \end{cases}$$

### 3.2.3 How to use interface of Configuration Module

There are several configurations for users, which are shown in table 3.2.3.

	Add	Remove	Edit
Fuzzy Concept	✓	✓	✓
Fuzzy Negation	✓	✓	✓
Fuzzy Quantification	✓	✓	✓
Concept Functions	✓	✓	✓
Negation Functions	✓	✓	✓
Quantification Functions	✓	✓	✓

Table 1: Operations in Configuration Module



Figure 3: Interface of Configuration Module

As shown in figure 3, the existing fuzzy concepts, negations and quantifications are shown by clicking subtitles “Concepts”, “Negations” and “Quantifications”, respectively. In each page, users can click “Add New concept”, “Add New negation” and “Add New Quantification” to enter Adding pages. Users can click hyperlink “edit function” behind the fuzzy concept, negation or quantification you want to edit to enter the Editing page.

# Fuzzy Query Answering

Main Page

Load DB

Configuration

Query

ConceptsNegationsQuantification

## Add concept/concept function

Name

Attribute

+

Add concept function

✓

Save

Figure 4: Interface of adding new concept

As shown in figure 4, users can give the name of fuzzy concepts in “Name” edit box, and choose an attribute from “Attribute” selection box. For example, putting “big” in “Name” edit box and choosing “size” from “Attribute” selection box, in order to create a concept “big” over attribute “size”. User can continue to add concept functions for this fuzzy concept by clicking “Add concept function” in the case of adding functions for this fuzzy concept, then save both fuzzy concept and its functions.

# Fuzzy Query Answering

Main Page

Load DB

Configuration

Query

Concepts

Negations

Quantification

## Add negation/negation function

Name

✓ Save

+

Add negation function

Figure 5: Interface of adding new negation

As shown in figure 5, adding negation and adding quantification are the same, users only need to type the name of negation or quantification and click button “Save”. Before saving, users can also add functions of negation or quantification by clicking “Add negation function” or “Add quantification function”.

# Fuzzy Query Answering

[Main Page](#)
[Load DB](#)
[Configuration](#)
[Query](#)

[Concepts](#)
[Negations](#)
[Quantification](#)

## Edit quantification very

Continuous  $[[0,0.5]] \times$  [remove](#)  
Continuous  $[[0,0.5]] \times$  [remove](#)  
Continuous  $[[0.5,1]] 1-x$  [remove](#)

Domain type

Domain range

Expression

Add quantification function

Figure 6: Interface of editing quantification

For editing functions, it is the same for fuzzy concepts, negations, and quantifications. As shown in figure 6, users can click hyperlink “edit functions” behind the fuzzy concept, negation, quantification you want to edit. After entering the page of editing functions, users can remove the certain existing functions by clicking the hyperlink “remove”. In order to add more functions, users can choose the function type from selection box “Domain type”, and type function domain in edit box “Domain range”, then write function expression in edit box “Expression”, the last step is to click button “Add quantification function” in the case user edits quantification functions. The new function will be added to the list of functions on the left.

### 3.3 Query-Answering Module

Query-Answering Module offers users to create simple fuzzy queries or complex fuzzy queries by choosing negations, quantifications and fuzzy concepts which are defined in Configuration Modules.

#### 3.3.1 Fuzzy Query

In RFuzzy Framework, we define the fuzzy query as a pair  $\langle A, v \rangle$ , where  $A \in TB_{\Pi, \Sigma, V}$  and  $v$  is either a “new” variable that represents the initially unknown truth value of  $A$  or it is a concrete value  $v \in [0, 1]$  that is asked to be the truth value of  $A$ . We extend this simple query into complex one.

**Definition 3.7. (Complex fuzzy query).**

$$Answer(\vec{t}, v) \xleftarrow{c, F_c} F(p_1(\vec{t}_1, v_1), \dots, p_m(\vec{t}_m, v_m))$$

where  $p_i$ s are predicates in RFuzzy program  $P$ ,  $Answer$  is a predicate which never appears in program  $P$ ,  $v_i$  and  $v$  could be unknown truth value for their associated atoms or concrete value  $v$  assigned to their atoms.

By introducing quantifiers into RFuzzy framework, it could be used to enhance the expressivity of the query, which is defined as follow,

**Definition 3.8. (Simple fuzzy query with negation and quantification).**

$$\langle A, v \rangle$$

where  $A$  is an atom  $q(x_1, \dots, x_n)$ .  $q$  is represented as a regular expression,

$$q = (Negation|Quantification)^* Predicate$$

**Definition 3.9. (Complex fuzzy query with negation and quantification).**

$$Answer(\vec{t}, v) \xleftarrow{c, F_c} F(q_1(\vec{t}_1, v_1), \dots, q_m(\vec{t}_m, v_m))$$

where  $Answer$  is a predicate that never appears in program  $P$ ,  $q_i$  is represented as a regular expression,

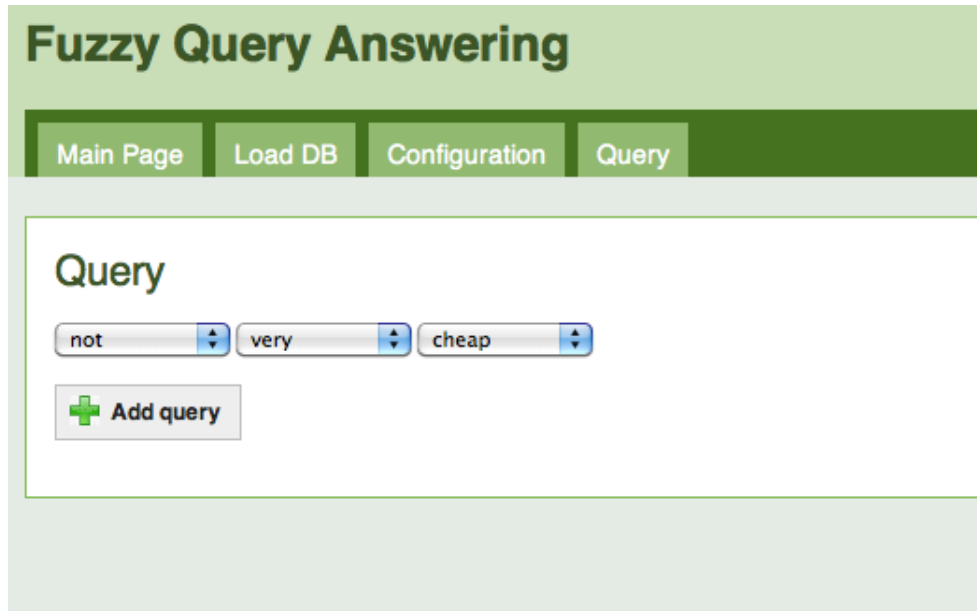
$$q_i = (Negation|Quantification)^* Predicate$$

**Example 3.10.** A set of negations is  $N = \{not, seldom\}$ , a set of quantifiers is  $Q = \{very, extremely\}$ , and a set of fuzzy concept is  $C = \{beautiful, clear\}$ . The simple query with negation and quantification can be represented as “not very beautiful girls?”, “seldom extremely clear statements?” and so on. A Complex query with negation and quantification is generated by joining those simple query together with fuzzy rules. The simplest complex query could be “not very beautiful and seldom extremely clear landscape?”, which is formalized in fuzzy logic as,

$$Answer(Landscape, V) \leftarrow \min \quad \begin{aligned} &not(very(beautiful(Landscape))), \\ &seldom(extremely(clear(Landscape))). \end{aligned}$$



### 3.3.2 How to use interface of Query-Answering Module



The screenshot displays the 'Fuzzy Query Answering' web interface. At the top, there is a green header bar with the title 'Fuzzy Query Answering'. Below the header is a navigation bar with four tabs: 'Main Page', 'Load DB', 'Configuration', and 'Query'. The 'Query' tab is currently selected. The main content area is titled 'Query' and contains three dropdown menus with the values 'not', 'very', and 'cheap' respectively. Below these menus is a button with a green plus icon and the text 'Add query'.

Figure 7: Interface of Query-Answering Module

As shown in figure 7, users can choose negation, quantification, and fuzzy concept from those three selection boxes. Users can add more queries by clicking button “Add query”, the previous query is listed on the right, and selection boxes are set to default empty ready for you to add more queries. By clicking button “Query”, the result will be generated and shown in a result page.

## 4 Conclusions

The goal of Fuzzy Query-Answering System is providing a user-friendly webInterface to make fuzzy configuration and fuzzy query over crisp database. Fuzzy Configuration encapsulates crisp database into a virtual fuzzy database, where the information is described in an imprecise and human recognizable way. Fuzzy Query offers users to make queries in a subset of natural language, which includes negation and quantification. To achieve the reasoning of fuzzy information, FQAS interacts with RFuzzy Framework, where the reasoning works out. The result is sent back to FQAS from RFuzzy Framework.

### 4.1 Disadvantages and Future work

The Fuzzy Query-Answering system as a user-friendly web interface, there are other options can be improved to make the system more user-friendly.

1. Loading File Module (LFM)

At present, most database are relational database, they have their own standard to store tables and database schema. The parser in LFM can extend to deal with more different of formats.

2. Configuration Module (CM)

Adding or editing functions can be built in a graphic approach, which is more human recognizable and convenient comparing with choosing domain type and typing domain range and function expression.

3. Query-Answering Module (QAM)

The result also can be presented as a graph, which is easier to be analyzed and chosen by users.

## References

- [APC02] Enric Trillas Ana Pradera and Tomasa Calvo. A general class of triangular norm-based aggregation operators: quasi-linear t-s operators. *International Journal of Approximate Reasoning*, 30(1):57–72, 2002.
- [ETC95] Susana Cubillo Enric Trillas and Juan Luis Castro. Conjunction and disjunction on  $([0,1], \leq)$ . *Fuzzy set and systems*, 72(2):155–165, 1995.
- [IK85] Mitsuru Ishizuka and Naoki Kanai. *Prolog-elf incorporating fuzzy logic*. Morgan Kaufmann Publishers Inc, 1985.
- [JM08] Pedro J.Morcillo and Gines Moreno. Programming with fuzzy logic rules by using the floper tool. In *RuleML '08: Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web*, pages 119–126, 2008.
- [KP00] Radko; Klement, Erich Peter; Mesiar and Endre Pap. *Triangular Norms*. Dordrecht: Kluwer, 2000.
- [Lee72] R. C. T. Lee. Fuzzy logic and the resolution principle. *the Association for Computing Machinery*, 19(1):119129, 1972.
- [LL90] Deyi Li and Dongbo Liu. A fuzzy prolog database system. 1990.
- [Llo87] John Wylie Lloyd, editor. Springer, 1987.
- [MHPCS10] Susana Munoz-Hernandez, Victor Pablos-Ceruelo, and Hannas Strass. Rfuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over prolog. 2010.
- [Mor06] Gines Moreno. Building a fuzzy transformation system. *Lecture Notes in Computer Science*, 3831:409–418, 2006.
- [ZM89] L.Ding Z.Shen and M.Mukaidono. Fuzzy resolution principle. *18th International Symposium on Multiple-valued Logic*, 5, 1989.