Víctor Pablos Ceruelo

NEGATIVE NON-GROUND QUERIES IN WELL FOUNDED SEMANTICS

Lisboa
2009

MCL

Víctor Pablos Ceruelo

2009

UNIVERSIDADE NOVA DE LISBOA
Facultade de Ciências e Tecnologia
Departamento de Informática

NEGATIVE NON-GROUND QUERIES IN WELL FOUNDED SEMANTICS

Por
Víctor Pablos Ceruelo

Dissertaçao apresentada na Facultade de Ciências e Tecnologia
da Universidade Nova de Lisboa para obtençao do grau de
Mestre em Computational Logic

Orientador: José Júlio Alves Alferes

Lisboa

2009

Abstract

To my girlfriend,
Sara,
with love.

CONTENTS

LIST OF FIGURES


Erasmus Mundus

Facultad de Informática

Universidad Politécnica de Madrid

European Master In Computational Logic

Master Thesis

A Sound And Efficient Approach For The Similarity Concept In Fuzzy Logic

Submitted by: Sinan Eğilmez

Supervisor: Susana Muñoz Hernández
Co-supervisor: Victor Pablos Ceruelo

October 2012

ABSTRACT

Real world applications often come with imprecise, uncertain or incomplete information. Fuzzy logic was born as a consequence of the incapability of classical two-valued logic systems when dealing with such domains.

The concept of similarity adds another layer to the field, as it introduces a fuzzy domain not only for the concepts, but also concerning the relations between concepts. This feature results into many means for flexible querying on any type of knowledge bases. For that reason, there has been a renewed interest in *Similarity-Based Logic Programming* over the last decade which has proved to be fruitful.

In this research we propose a new methodology concerning evaluating the similarity proximities in fuzzy logic domain, which eliminates all shortcomings of the preceding approaches. Moreover we introduce a framework for approximating the precision of our method by presenting a fully automized algorithm. We demonstrate the value of our work via displaying the accuracy of the method on real-world scenarios.

ACKNOWLEDGEMENTS

This work would not have been possible without the contribution of several people.

Firstly I would like to thank my supervisors Susana Muñoz-Hernández and Victor Pablos Ceruelo for their extreme support, understanding and patience. I really appreciate the fact that they have allowed me to work on a subject, which I sincerely enjoyed and felt passionate about it through the whole process.

I would like to extend a special thanks to Victor since without his unparalleled input, this work would never be even remotely as good as it's final state. Sometimes in life lady luck smiles at you, and you meet people who are willing to help you with everything they got, regardless of the fact that they do not owe you a penny. Victor is one such unique person and I'm so glad for having the privilege of working with him.

I'm highly grateful to a list of people starting with prof. Hölldobler, Dr.Wernhard, Sylvia Wunsch, Paloma Vivas, Julia Koppenhagen and many others for organizing such an amazing program in EMCL. I have spent two wonderful years in two beautiful countries, which I will cherish for the rest of my life.

It was such a pleasure spending two years with my *compadres*, Alex and Alvaro. Because of you guys I never had a dull moment during our studies.

I want to thank my dear friends Deniz, Hakan, Serdar and the *projedeyim* gang for making my casual breaks from the thesis study sessions something I eagerly look forward to.

I owe a huge debt to my high-school teacher Erol Baksi for showing me the path *from zero to hero*. I consider him as a great source of inspiration in my life and can only dream of being able to follow his footsteps one day.

Last but not least, I must state that I feel immensely indebted to my beloved family, but especially to my mother and my brother Selim for always being there when I needed them, for never having stopped believing in me and for showing me that for every dark night, there's a brighter day.

CONTENTS

CHAPTER 1

INTRODUCTION

In the field of databases and information retrieval, there is an ever increasing demand for systems able to deal with flexible queries and answers. For instance with the limited capabilities of today's search engines, when one searches for *fast car*, the results are websites over the internet with texts which include the words *fast* and *car*. The search engines are simply not proficient enough to observe from one source that a particular car has a maximum speed of 350 Km/h, and thus it should be returned as an answer of the query. Moreover they do not possess the means of detecting not exact, but closely related answers. Here when we say related, we do not mean lexical similarity, but indeed meaningful semantic likelihood. So as another simple example we might state that when queried for *red car*, no search engine lists *orange cars* as a potential point of interest. What we would want from them is to tell us that this is not the exact answer, but still one which should be taken into consideration. Under this topic, dealing with similarity relations is a subject which requires utmost attention; where a similarity relation stand for the degree of affinity between two entities of the concerning domain.

Fuzzy Logic is used to represent vague information in the real world. In order to draw the similarity between objects in real world, we introduce similarity between predicates into Fuzzy Logic. Similarity itself is a fuzzy concept, which makes fuzzy logic suitable and appropriate for representing similarity.

There are different approaches which amalgamates *Logic Programming* with concepts coming from *Fuzzy Logic*. This type of works belong to logic programming classes such as *Fuzzy Logic Programming* [GMHV04], *Qualified Logic Programming* [CRARD08] and *Similarity-Based Logic Programming* [Ses02].

One intriguing element of the latter class is the *Bousi∼ Prolog* [JIRM11]. Briefly we may say that *Bousi∼ Prolog* belongs to the *Similarity-Based Logic Programming* class, and it replaces the syntactic unification mechanism of classical SLD-resolution by a fuzzy unification algorithm based on fuzzy binary relations on a syntactic domain. This algorithm provides a weak most general unifier as well as a numerical value, called the *approximation degree*. The approximation degree represents the truth degree associated with the computed instance(query). The result is an opera-

tional mechanism, called Weak SLD-resolution, which differs from other approaches in some aspects, based exclusively on similarity relations. Very crudely, the framework utilizes an algorithm which firstly annotates each formula in the rule set with a truth degree equal to 1. The rest of the formulas are annotated with their corresponding approximation degree. In case that several formulas of set generate the same approximate formula, with different approximations degrees, the one with the least degree is taken as annotation. The resulting set consist the core concepts of model and logical consequence of the particular similarity relation degree which is chosen at the beginning of the method.

On the other hand, a very promising representative of the first class is the *RFuzzy* framework, a Prolog-based tool for representing and reasoning with fuzzy information. In a nutshell the advantages of the framework in comparison to other tools in the same field of research are its easy, user-friendly syntax, and its expressivity through the availability of default values and types. [MHPCS10]

The similarity concept has been recently investigated in the *Rfuzzy* framework. As of today, in *Rfuzzy* framework there are two existing methods for measuring the degree of similarity between two fuzzy predicates. Namely: *Interpretation Based* and *Structure Based* methods. Crudely, the former obtains the similarity between fuzzy predicates by the comparison between their interpretations.Whereas the latter method considers the structure of fuzzy predicates, in other words the way that fuzzy rules define fuzzy concepts. [Lu11]

Even though these approaches are sound methodologies they are not without some flaws:

- As mentioned above, Interpretation Based Method(*IBM*) is a very naive method which only utilizes the commonality of the attributes of the predicates'. It can only work on a single layer of subconcepts thus does not have the means of tackling with complex definitions.

- Unlike *IBM*, Structure Based Method(*SBM*) makes use of the structural property by constructing the predicate tree via taking the head of a fuzzy rule as the root of the tree and by branching the tree regarding the body of the rule. Here one point of concern is that, the construction of the tree is recursive but not the evaluation of the similarity degree between the predicates. As a result of this the trees need to be structurally equivalent for the algorithm to be able to execute. This hole in the algorithm is patched by introducing identity predicates when one tree is lacking internal nodes or leaf nodes compared to the other one. But as Lu also confirms in [Lu11], this causes a "distortion" effect on the final evaluation. Especially in cases such as where the branching factors differ by a big margin or when one tree is highly unbalanced, this effect would be clearly apparent.

With these shortcomings in mind, we believe a new approach which would handle the mentioned problems would carry significant importance. Very briefly the idea is firstly defining a generic entity matching problem(whether it being terms or predicates) mathematically as a graph problem and then introduce computational methods

utilizing properties of graphs, to identify structural resemblance between parts of the graphs. That would be followed by fuzzyfying the process by introducing fuzzy logic terms on this generic framework. Lastly a methodology regarding computing the confidence value of the result of the process is constructed.

In this thesis, preliminaries are displayed in Chapter 2, where lattice theory, fuzzy set/relation, Fuzzy Logic and a brief introduction of RFuzzy framework are presented. In Chapter 3 we observe the *SBM* method [Lu11] in detail and demonstrate its shortcomings. Chapter 4 depicts the new methodology we introduce in to the field. Since this methodology computes results which are not hundred percent valid in the cases of knowledge bases with incomplete information, in Chapter 5 we propose a method for computing the credibility values of fuzzy similarity relations. With the last chapter we wrap up the research and show on going research topics. The contribution in this work fixes the existing gaps in Lu's approach [Lu11] and defines the roots for a much more ambitious project, as mentioned getting not only the results for *red car* but the ones for *orange car*, too.

CHAPTER 2

PRELIMINARIES

2.1 Background

The section presents the needed background knowledge is introduced for clarifying the foundations of fuzzy logic. Lattice theory [DP03] is the key to bridge the corresponding relation between fuzzy set/relation and fuzzy logic, and it is illustrated in section 2.1.1. Fuzzy set/relation and fuzzy logic are represented in sections 2.1.2 and 2.1.3, respectively.

2.1.1 Lattice Theory

The definition of lattice is given by starting from set theory, followed by some important theorems and propositions as well as their proofs. Those theorems and propositions are used in chapter 3 to bridge fuzzy set/relation and fuzzy logic.

Definition 2.1.1
Relation over multiple sets. Let S_i be sets, where $i \in [1, n]$, a relation R over all the sets S_i is a subset of **Cartesian product** of the sets S_i, that is, $R \subseteq \Pi_{i \in [1,n]} S_i$.

Definition 2.1.2
Relation over single set. A relation R with arity n over a set S is a subset of **Cartesian product** S^n, that is, $R \subseteq S^n$.

Definition 2.1.3
Binary relation over single set. A binary relation R over a set S is subset of **Cartesian product** $S \times S$, that is, $R \subseteq S \times S$.

With respect to the binary relation, there are several interesting properties over it, such as reflexivity, antisymmetry, transitivity. Let R be a binary relation over S, and suppose x, y, z are elements in S.

Definition 2.1.4
Partial order. A binary relation R over S is called a partial order iff it satisfies the properties below,

- **Reflexivity** $\forall x$, if $x \in S$, then the pair $(x,x) \in R$
- **Antisymmetry** if $(x,y) \in R$, and $(y,x) \in R$, then $x = y$
- **Transitivity** if $(x,y) \in R$ and $(y,z) \in R$, then $(x,z) \in R$

Definition 2.1.5
Partial ordered set. A partial ordered set is a pair $(U, \preceq)$, where U is a set, and $\preceq$ is a partial order over U.

Example 2.1.6
$(\mathbb{N}, \leq)$ is partial order set, and so is $(\mathcal{P}(U), \subseteq)$, where $\mathcal{P}(U)$ is the power set of U, that is, $\mathcal{P}(U)$ is the set of all subsets of U.

Definition 2.1.7
Supremum. For subsets S of arbitrary partially ordered sets $(P, \preceq)$, a supremum or least upper bound of S is an element u in P such that

- u is a upper bound of S, notated as,
  $\forall x \in S.\ x \preceq u$
- u is a least upper bound of S, notated as,
  $\forall v \in P$ such that $x \preceq v$ for all x in S, it holds that $u \preceq v$

The special case is that a subset of $(P, \preceq)$ only with two elements, the supremum of such set is notated as $sup\{x,y\}$, which is used for the operations over fuzzy sets as their **join** and written as $x \vee y$.

Definition 2.1.8
Infimum. Formally, the infimum or greatest lower bound of a subset S of a partially ordered set $(P, \preceq)$ is an element m of P such that

- m is a lower bound of S, notated as,
  $\forall x \in S.\ m \preceq x$
- m is a greatest lower bound of S, notated as,
  $\forall v \in P$ such that $v \preceq x$ for all x in S, it holds that $v \preceq m$

The special case is that a subset of $(P, \preceq)$ only with two elements, the supremum of such set is notated as $inf\{x,y\}$, which is used for the operations over fuzzy sets as their **meet** and written as $x \wedge y$.

Definition 2.1.9
Lattice. A lattice is a partially ordered set $(P, \preceq)$ if every pair of elements of P has a sup and an inf in P.

Example 2.1.10
The partially ordered set $(\mathcal{P}(U), \subseteq)$ is a lattice. The sup
(supremum) of two elements in $\mathcal{P}(U)$ is their union, and inf(infimum) is their intersection. The interval $[0,1]$ is a lattice, with $inf\{x,y\} = min\{x,y\}$ and $sup\{x,y\} = max\{x,y\}$.

From the definition of lattice, if $(U, \preceq)$ is a lattice, then it comes equipped with the two binary operations **join**($\vee$) and **meet**($\wedge$) described above. From either of these binary operations, $\preceq$ can be reconstructed. In fact, $a \preceq b$ if and only if $a \wedge b = a$ if and only if $a \vee b = b$. These two binary operations satisfy a number of properties exposed in Theorem 2.1.11.

Theorem 2.1.11
If $(U, \preceq)$ is a lattice, then for all $a, b, c \in U$.

- $a \vee a = a$ and $a \wedge a = a$. ($\vee$ and $\wedge$ are **idempotent**.)
- $a \vee b = b \vee a$ and $a \wedge b = b \wedge a$. ($\vee$ and $\wedge$ are **commutative**.)
- $(a \vee b) \vee c = a \vee (b \vee c)$ and $(a \wedge b) \wedge c = a \wedge (b \wedge c)$. ($\vee$ and $\wedge$ are **associative**.)
- $a \vee (a \wedge b) = a$ and $a \wedge (a \vee b) = a$. (These are the **absorption identities**.)

Theorem 2.1.12
If U is a set with binary operations $\vee$ and $\wedge$ which satisfy the properties of Theorem 2.1.11, then defining $a \preceq b$ if $a \wedge b = a$ makes $(U, \preceq)$ a lattice whose **sup** and **inf** operations are $\vee$ and $\wedge$.

Proof. We first show that $a \wedge b = a$ **iff** $a \vee b = b$. Thus defining $a \preceq b$ if $a \wedge b = a$ is equivalent to defining $a \preceq b$ if $a \vee b = b$.

- To prove "If $a \wedge b = a$, then $a \vee b = b$."

  If $a \wedge b = a$, then

  $$\begin{array}{rcll} a \vee b & = & (a \wedge b) \vee b & \\ & = & b \vee (b \wedge a) & \textbf{(commutative)} \\ & = & b & \textbf{(absorption identities)} \end{array}$$

- To prove "If $a \vee b = b$, then $a \wedge b = a$."

  If $a \vee b = b$, then

  $$\begin{array}{rcll} a \wedge b & = & a \wedge (a \vee b) & \\ & = & a & \textbf{(absorption identities)} \end{array}$$

$(U, \vee, \wedge)$ with the definition of $\preceq$ that $a \preceq b$ iff $a \wedge b = a$ makes $(U, \preceq)$ into a partial order set. We specify this point from the definition of partial order set and describe $\preceq$ with three properties over set $U \times U$.

- *reflexivity*

  It shows that $a \wedge a = a$ by **idempotent** rule, then from the definition of $a \preceq b$, which is "if $a \wedge b = a$, then $a \preceq b$", $a \preceq a$ is drawn.

- *antisymmetry*

  If $a \preceq b$, then $a \vee b = a$ and $b \preceq a$ implies $b \vee a = b$ by the definition of $\preceq$. since $a \vee b = b \vee a$ by **commutative**, it is drawn that $a = b$. Thus, If $a \preceq b$ and $b \preceq a$, then $a = b$.

- *transitivity* If $a \preceq b$ and $b \preceq c$, since $a \preceq b$ implies $a \wedge b = a$, while $b \preceq c$ implies $b \wedge c = b$, Then,

$$\begin{aligned} a \wedge c &= (a \wedge b) \wedge c \\ &= a \wedge (b \wedge c) \quad \textbf{(associative)} \\ &= a \wedge b \\ &= a \end{aligned}$$

  $a \wedge c = a$ implies that $a \preceq c$. Therefore, if $a \preceq b$ and $b \preceq c$, then $a \preceq c$.

The binary relation $\preceq$ built on $\vee$ and $\wedge$ satisfies *reflexivity*, *antisymmetry* and *transitivity*, which makes $\preceq$ a partial order. Thus, $(U, \preceq)$ is partial order set.

Next, to show the existence of sups, we claim that $sup\{a,b\} = a \vee b$. The proof is as follows, since we define $a \preceq b$ if $a \wedge b = a$, and $a \wedge (a \vee b) = a$ by the absorption identities, then $a \preceq a \vee b$ is drawn. While, $b \wedge (a \vee b) = b \wedge (b \vee a) = b$ by commutative and absorption rules, therefore $b \preceq a \vee b$. Since $a \preceq a \vee b$ and $b \preceq a \vee b$, $a \vee b$ is an upper bound of a and b. For any upper bound x, $a \vee x = x$ and $b \vee x = x$. $x = x \vee x = (a \vee x) \vee (b \vee x) = (a \vee b) \vee x$, makes $a \vee b \preceq x$. Thus, $a \vee b$ is a smallest upper bound of a and b, that is, $sup\{a,b\} = a \vee b$.

Claiming that $inf\{a,b\} = a \wedge b$, it will be proved as the existence of infs. $(a \wedge b) \wedge a = a \wedge (a \wedge b) = (a \wedge a) \wedge b = a \wedge b$, by the definition of $\preceq$, $a \wedge b \preceq a$. While $(a \wedge b) \wedge b = a \wedge (b \wedge b) = a \wedge b$, implies $a \wedge b \preceq b$. Since $a \wedge b \preceq a$ and $a \wedge b \preceq b$, $a \wedge b$ is a lower bound of a and b. For any lower bound x, $x \wedge a = x$ and $x \wedge b = x$. $x = x \wedge x = (x \wedge a) \wedge (x \wedge b) = x \wedge (a \wedge b)$ implies $x \preceq a \wedge b$, therefore, $a \wedge b$ is the largest lower bound of a and b, that is, $inf\{a,b\} = a \wedge b$.

□

The Theorem 2.1.12 shows different approaches to define lattice, as well as the point of view. A lattice could be seen as a partial order in which every pair of elements has an inf and a sup, or a set with a pair of operations which satisfy the properties mentioned in Theorem 2.1.11. In general case, $(U, \preceq)$ is a lattice and so is $(U, \vee, \wedge)$ with the properties in Theorem 2.1.11.

Some pertinent additional properties of a lattice $(U, \preceq)$ may have,

- *property 1:* 0 and 1 are **identities** for $\vee$ and $\wedge$ respectively.

  There is an element 0 in U such that $0 \vee a = a$ for all $a \in U$. There is an element $1 \in U$ such that $1 \wedge a = a$ for all $a \in U$.

- *property 2:* Each element in U has a **complement**

  Let U have identities, and for each element a in A, there is an element a' in U such that $a \wedge a' = 0$ and $a \vee a' = 1$.

- *property 3:* The binary operations $\vee$ and $\wedge$ **distribute** over each other.

  $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ and $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$.

- *property 4:* Every subset T of U has a sup.

- *property 5:* Every subset T of U has an inf.

If a lattice has an identity for $\vee$ and an identity for $\wedge$, then it is a **bounded lattice**. A bounded lattice satisfying *property 2* is a **complemented lattice**. A lattice satisfying both distributive laws for $\vee$ and $\wedge$ in *property 3* is a **distributive lattice**. A bounded distributive lattice that is complemented is a **Boolean lattice**, or **Boolean Algebra**. A lattice satisfying *property 4* and *property 5* is a **complete lattice**.

The interval $[0,1]$ is a complete lattice, and so is $(\mathcal{P}(U),\subseteq)$.

There is an operation denoted as $'$ with the following properties,

1. $(x')' = x$
2. $x \leq y$ implies that $y' \leq x'$

The operation " ' " on a bounded lattice is called an **involution**, or **duality**. If $'$ is an involution, the equations

$$(x \vee y)' = x' \wedge y'$$
$$(x \wedge y)' = x' \vee y'$$

are called the **De Morgan laws**, and may or may not hold. A system satisfying **De Morgan laws** $(V,\vee,\wedge,',0,1)$ is a **De Morgan algebra**.

The lattice $([0,1],\leq)$ is a De Morgan algebra, where its $'$ operation is $[0,1] \rightarrow [0,1] : x \rightarrow 1-x$. The lattice $(\mathcal{P}(U),\subseteq,',\emptyset,U)$ is also a De Morgan algebra, with $'$ as a complement operation on a subset of U.

Theorem 2.1.13
Let $(V,\vee,\wedge,',0,1)$ be a De Morgan algebra and let U be any set. Let f and g be mappings from U into V. We define

- *Rule 1:* $(f \vee g)(x) = f(x) \vee g(x)$
- *Rule 2:* $(f \wedge g)(x) = f(x) \wedge g(x)$
- *Rule 3:* $f'(x) = (f(x))'$
- *Rule 4:* $0(x) = 0$
- *Rule 5:* $1(x) = 1$

Let V^U be the set of all mappings from U into V. Then $(V^U,\vee,\wedge,',0,1)$ is a De Morgan algebra. If V is a complete lattice, then so is V^U.

Proof. Let f, g and h be arbitrary mappings from U to V, so they are elements in V^U. The lattice could be considered in two different ways, which are introduced in Theorem 2.1.12. One is, a lattice could be seen as a partial order in which every pair of elements has a **sup** and a **inf**. The other is, a lattice is viewed as a set with a pair of operations satisfying the properties in Theorem 2.1.11, where $\vee$, $\wedge$ have properties **idempotency**, **commutativity**, **associativity** and **identity absorption**. In order to prove that V^U is a complete lattice, there are two points to be demonstrated, one is that $(V^U,\vee,\wedge)$ is a lattice and the other is that each subset of V^U has an inf and a sup.

- $(V^U, \vee, \wedge)$ is a lattice.

  - **idempotency**
    $(f \vee f)(x) = f(x) \vee f(x)$ by **rule 1**, $f(x) \in V$ since $f : U \to V$, and $(V, \vee, \wedge)$ is a complete lattice. Therefore, $f(x) \vee f(x) = f(x)$ by using idempotency over V. Thus, $(f \vee f)(x) = f(x)$, that is idempotent holds in V^U. $(f \wedge f)(x) = f(x)$ could be proved in a similar way.
  - **commutativity**
    $$\begin{array}{rcll} (f \vee g)(x) & = & f(x) \vee g(x) & \textbf{(Rule 1)} \\ & = & g(x) \vee f(x) & \textbf{(Commutative Rule} \text{ over } V) \\ & = & (g \vee f)(x) & \textbf{(Rule 1)} \end{array}$$
    $(f \wedge g)(x) = f(x) \wedge g(x)$ could be proved in the same way.
  - **associativity**
    $$\begin{array}{rcll} ((f \vee g) \vee h)(x) & = & (f \vee g)(x) \vee h(x) & \textbf{(Rule 1)} \\ & = & (f(x) \vee g(x)) \vee h(x) & \textbf{(Rule 1)} \\ & = & f(x) \vee (g(x) \vee h(x)) & \\ & & \quad (\ \textbf{Associative Rule} \text{ over } V) & \\ & = & f(x) \vee ((g \vee h)(x)) & \textbf{(Rule 1)} \\ & = & (f \vee (g \vee h))(x) & \textbf{(Rule 1)} \end{array}$$
    $((f \wedge g) \wedge h)(x) = (f \wedge (g \wedge h))(x)$ could be proved in the same way.
  - **absorption identities**
    $$\begin{array}{rcll} (f \vee (f \wedge g))(x) & = & f(x) \vee (f(x) \wedge g(x)) & \textbf{(Rule 1)} \\ & = & f(x) & \\ & & (\ \textbf{Absorption identities} \text{ over } V) & \end{array}$$
    $(f \wedge (f \vee g))(x) = f(x)$ could be proved in the same way.

- Each subset of V^U has an inf and a sup.

  Since $(V^U, \vee, \wedge)$ is a lattice, then every pair of elements in V^U has an inf and a sup. It implies that each subset of V^U also has an inf and a sup, the proof can be done inductively, and can be found in [DP03].

□

2.1.2 Fuzzy Set and Fuzzy Relation

The mathematical modeling of fuzzy concepts was presented by Zadeh in 1965, and his approach is described here. His intention is to represent the meaning in natural language as a matter of degree. For instance, for a proposition such as "Mary is young.", it is not always possible to be asserted as either true or false. It depends on the definition of concept "young", if "young" is given as a set

$$Young = \{x | x \in [0, \infty), x < 23\}$$

and Mary's age is given as 21, then the proposition "Mary is young" is true under the definition of "young" above. It seems that classical set theory solves the problem

of representing the imprecise information such as 'young' in the the example above. However, 18 and 20 year olds are young, but with different degrees: 18 is younger than 20. This suggests that the membership in 'Young' concept should not be on a 0 or 1 basis, but rather on 0 to 1 scale, that is, the membership should be an element of the interval [0,1].

For representation of membership with degree, fuzzy set is defined in the same way as the definition of crisp set.

Definition 2.1.14
Crisp Set. Let U be the universal set, an ordinary subset A of U is determined by its **indicator function**, or **characteristic function** χ_A defined by

$$\chi_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

The indicator function of a subset A of the universal set U specifies whether an element is in A or not. Only two values can be taken for the characteristic function, 0 and 1. The characteristic function could be generalized as $\chi_A : U \to \{0,1\}$.

According to the definition above, crisp set only has 0, 1 as there membership degree. However, in fuzzy set, the degree value is a real number in $[0,1]$.

Definition 2.1.15
Fuzzy Set. Let U be the universal set, which includes all the elements in our interested domain. A **fuzzy subset** A of the universal set U is a function $\mu_A : U \to [0,1]$. It is common to refer to a fuzzy subset simply as a **fuzzy set**.

The image of the fuzzy function μ_A contains two values 0 and 1, which correspond to the image of crisp subset of U. Therefore, crisp subsets are considered as special cases of fuzzy subsets.

It is customary in fuzzy literature to represent fuzzy set in both notations, A and μ_A. The first one is called 'linguistic label', which indicates the concept of interested domain. For example, Let A be a set of young people. Then, the $\mu_A : U \to [0,1]$ is called fuzzy function, representing the degree of youngness that has been assigned to each member in U.

Suppose that all the numbers which are specified as ages belong to U, then a value in $[0,1]$ is assigned to each number in U. For example, $\mu_{young}(18) = 0.85$, $\mu_{young}(20) = 0.81$, then "18 is younger then 20" has its semantic meaning under μ_{young} or concept 'young', which is $\mu_{young}(18) > \mu_{young}(20)$.

Operations on fuzzy set

Following the definition above, a crisp subset A of universal set U can be represented by a function $\chi_A : U \to \{0,1\}$, while a fuzzy subset $\tilde{A}$ of universal set U is indicated as a function $\mu_{\tilde{A}} : U \to [0,1]$. On the subsets of U, whether they are crisp or fuzzy, the operations of union, intersection, and complement are given as follows.

The original definition is given by the rules

$$A \cup B = \{x | x \in A \text{ or } x \in B\}$$
$$A \cap B = \{x | x \in A \text{ and } x \in B\}$$
$$A^{'} = \{x \in U | x \notin A\}$$

From the point of view of function, the operations over crisp subsets are:

$$\chi_{A \cup B}(x) = max\{\chi_A(x), \chi_B(x)\} = \chi_A(x) \vee \chi_B(x)$$
$$\chi_{A \cap B}(x) = min\{\chi_A(x), \chi_B(x)\} = \chi_A(x) \wedge \chi_B(x)$$
$$\chi_{A'}(x) = 1 - \chi_A(x)$$

Naturally, by the membership function, these operations over fuzzy subsets of U are extended as,

$$\mu_{\tilde{A} \cup \tilde{B}}(x) = sup\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} = \mu_{\tilde{A}}(x) \vee \mu_{\tilde{B}}(x)$$
$$\mu_{\tilde{A} \cap \tilde{B}}(x) = inf\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} = \mu_{\tilde{A}}(x) \wedge \mu_{\tilde{B}}(x)$$
$$\mu_{\tilde{A'}}(x) = 1 - \mu_{\tilde{A}}(x)$$

The syntax $\vee$ and $\wedge$ are the operations over sets, which are mentioned in section 2.1 Lattice. Note that they are different from the logic notation used for 'or' and 'and'.

Fuzzy Relation

Relations, or associations among objects, are of fundamental importance in the analysis of real-world system. Since the real-world is full of vague and uncertain information, the fuzzy relation plays an important role to represent it.

The classical relation has been defined in the lattice section. In this subsection, the fuzzy relation will be generalized from the crisp one.

Definition 2.1.16
Fuzzy Relation. An **n-ary fuzzy function relation in a set** $V = U_1 \times U_2 \times ... \times U_n$ is a fuzzy subset R of V, that is, a function $R : V \to [0,1]$. If the sets U_i are identical, say U, then R is an **n-ary fuzzy relation on** U.

2.1.3 Fuzzy Logic

Syntax

Let Σ be a set of function symbols, V be a set of variables and Π be a set of predicate symbols. Terminology such as terms, atoms are formally defined.

Definition 2.1.17
Term

- t is a term, if $t \in V$
- $f(t_1, ..., t_n)$ is a term, if $f \in \Sigma$ and t_i are terms, where $i \in [1, n]$.

- nothing else is term.

A special case is that f's arity is 0, then the term is called *constant*. The set of all the terms is called **term universe**, notated as $TU_{\Sigma,V}$.

Definition 2.1.18
Atom $p(t_1,...,t_n)$ is an atom if $p \in \Pi$ and t_i are terms.

A special case is that p's arity is 0, then the atom is called *proposition*. The set of all the atoms built under V, Σ, Π is called **term base**, notated as $TB_{\Pi,\Sigma,V}$.

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

Semantics

In classical logic, the interpretation of formula only could be 0 or 1, but in fuzzy interpretation, the real number $v \in [0,1]$ is assigned to an *atom*.

Let $\mathbb{T}$ be an interval $[0,1]$, which is a value pool the atoms will be assigned to. $(\mathbb{T},\leq)$ is a complete lattice as we know from lattice section. A *valuation* $\sigma : V \to \mathbb{HU}$ is an assignment of ground terms to variables. Each valuation σ uniquely constitutes a mapping $\hat{\sigma} : TU_{\Pi,\Sigma,V} \to \mathbb{HB}$ that is defined in the obvious way. A *fuzzy Herbrand interpretation* of a fuzzy logic is a mapping $I : \mathbb{HB} \to \mathbb{T}$ that assigns truth values to ground atoms.

For two interpretations I and J, we say I *is less than or equal to* J, written $I \sqsubseteq J$, iff $I(A) \preceq J(A)$ for all $A \in \mathbb{HB}$. Two interpretations I and J are equal, written $I = J$, iff $I \sqsubseteq J$ and $J \sqsubseteq I$. Minimum and maximum for interpretations are defined from $\preceq$ as usual. Accordingly, the infimum and supremum of interpretation are, for all $A \in \mathbb{HB}$, defined as $(I \sqcap J)(A) = min(I(A), J(A))$ and $(I \sqcup J)(A) = max(I(A), J(A))$.

2.2 RFuzzy Framework

RFuzzy framework is a Prolog-based tool for representing and reasoning with fuzzy information. In this section we present a brief overlook for the RFuzzy framework [MHPCS10] in this chapter to familiarize the reader with the syntax and semantics which we work on in similarity and quantification.

2.2.1 Introduction

Logic programming [Llo87] has been successfully used in knowledge representation and reasoning for decades. Indeed, world data in 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 constrains.

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).

2.2.2 Syntax

From Σ 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 Σ 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*.

Ω is a set of *many-valued connectives* and includes,

1. conjunctions $\&_1$, $\&_2$, ..., $\&_k$

2. disjunctions $\vee_1$, $\vee_2$, ..., $\vee_l$

3. implications $\leftarrow_1$, $\leftarrow_2$, ..., $\leftarrow_m$

4. aggregations $@_1$, $@_2$, ..., $@_n$

5. real numbers $v \in [0,1] \subset \mathbb{R}$. These connectives are of arity 0, that is, $v \in \Omega^{(0)}$.

While Ω 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 2.2.1
(fuzzy clauses). A fuzzy clause is written as,

$$A \overset{c,F_c}{\leftarrow} F(B_1, ..., B_n)$$

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 $F_c \in \{\&_1, ..., \&_k\} \subset \Omega^{(2)}$ 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 2.2.2

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

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 2.2.3
(default value declaration). A *default value declaration* for a predicate $p \in \Pi^{(n)}$ is written as

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

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

Example 2.2.4

$$default(nice - weather(X)) = 0.5$$

Definition 2.2.5
(type declaration). Types are built from terms $t \in \mathbb{HU}$. A *term type declaration* assigns a type $\tau \in \mathcal{T}$ to a term $t \in \mathbb{HU}$ and is written as $t : \tau$. A *predicate type declaration* assigns a type $(\tau_1, ..., \tau_n) \in \mathcal{T}^n$ to a predicate $p \in \Pi^n$ and is written as $p : (\tau_1, ..., \tau_n)$, where τ_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 2.2.6
Suppose that the set of types is $\mathcal{T} = \{City, Continent\}$, here are some examples of *term type declaration* and *predicate type declaration*.

$$madrid : City, sydney : City$$

$$africa : Continent, america : Continent$$

$$nice - weather : (City)$$

$$city - continent : (City, Continent)$$

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 2.2.7
(well-typed). A ground atom $A = p(t_1, ..., t_n) \in \mathbb{HB}$ is *well_typed* with respect to T **iff** $p : (\tau_1, ..., \tau_n) \in T$ implies that the term type declaration $(t_i : \tau_i) \in T$ for $1 \leq i \leq n$.

A ground clause $A \overset{c,F_c}{\leftarrow} F(B_1, ..., B_n)$ 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 2.2.8
(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

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

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

2.2.3 Semantics

A *valuation* $\sigma : V \rightarrow \mathbb{HB}$ is an assignment of ground terms to variables. Each valuation σ 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 2.2.9
(Model). Let $P = (R, D, T)$ be a fuzzy logic program.

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

$$I \models A \overset{c,F_c}{\leftarrow} F(B_1, ..., B_n)$$

iff for all valuations σ:

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

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

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 σ:

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

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

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

iff for all valuation σ :

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

where φ_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 $I \models D$.

CHAPTER 3

THE NEED FOR A NEW PROPOSAL TO EVALUATE SIMILARITY BETWEEN FUZZY PREDICATES DEFINED IN A FUZZY PROGRAM

In chapter 1 we have mentioned about some existing works in the field of similarity concept in Fuzzy Logic, and also the *SBM* method [Lu11] has been highlighted as an intriguing work in the field. In this chapter the *SBM* method is observed in detail, and the layout of the section is as follows:

In Structured Based Measurement, a tree is built for each predicate, so comparing two predicates is indeed comparing two trees. In the light of that, section 3.1.1 introduces the approach for building a tree for predicate. In section 3.1.2, the algorithm to obtain similarity between two trees is described in detail. Then the shortcomings of the approach are displayed with their corresponding examples in section 3.2.

3.1 Structured Based Measurement

3.1.1 Predicate tree

Firstly, the question of "how to build a tree for a predicate in *RFuzzy* Program" is the main focus. For convenience, the corresponding tree for the predicate is called as a *predicate tree*. The first step is transforming an *RFuzzy* Program, to obtain the subset of program which is used to create the predicate tree in section 3.1.1. The approach of constructing predicate trees from subset of program is presented in section 3.1.1. In SBM method, to be able to compare two predicate trees, the structure of the concerning trees should be the same. Moreover they ought to preserve the embedded meanings. To ensure this, the concept "equivalent trees" is introduced in section 3.1.1. According to "equivalent trees", the predicate tree could be constructed with *identities*, which is claimed to maintain the semantic meaning but reform the structure. This is described in section 3.1.1.

Transforming RFuzzy program

In an *RFuzzy program* $P = (R, D, T)$, R is a set of *fuzzy clauses*, D is a set of *default value declarations* and T is a set of *type declarations*. In order to construct a tree for each predicate in P, there is the need of filtering and reforming some parts of P. This procedure is called *transformation*, in which two steps are taken, one is filtering and the other is reforming. In this section, the details *how* and *why* are presented [Lu11].

The result of first step *filtering* is a tuple $P' = (R', D', T')$, which is generated from P by following constrains,

1. Refined R'

   R as a set of *fuzzy clauses*, includes fuzzy clauses, which are written as,

   $$A \xleftarrow{c,F_c} F(B_1, ..., B_n)$$

   where $A \in TB_{\Pi,\Sigma,V}$ is called the head, $B_1, ..., B_n \in TB_{\Pi,\Sigma,V}$ is called the body, $c \in [0,1]$ is the credibility value, and $F_c \in \{\&_1, ..., \&_k\} \subset \Omega^{(2)}$ and $F \in \Omega^{(n)}$ are connectives symbols. F_c is to combine the credibility of the rule and the truth value of the body. F is to combine the truth values of the subgoals in the body.

   A *fuzzy fact* is a special case where $c = 1$, F_c is the usual product of real numbers " . " , $n = 0$, $F \in \Omega^{(0)}$. It is written as

   $$A \longleftarrow v$$

   where c and F_c are omitted.

   The similarity between predicates is a relevant value of comparison between two predicates, rather than the absolute value described in *fuzzy facts*. Therefore, *fuzzy facts* is out of consideration of similarity between predicates.

   As a result, R' is a set of all *fuzzy clauses*, but not *fuzzy facts*, notated as, $R' = R \backslash R_{facts}$.

2. Refined D'

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

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

   where $\delta_i \in [0,1]$ for all i. The φ_i are first-order formulas restricted to terms in p from TU_{Σ,V_p}, the predicates $=$ and $\neq$, the symbol true and the junctors $\wedge$ and $\vee$ in their usual meaning, which are *'and'*, *'or'*.

   There is a special case called *unconditional* default value declaration where $m = 1$ and $\varphi_1 = true$ in the default value declarations. In this case, the predicate p/n is not related to any other predicates, and therefore it is out of consideration in similarity.

   All in all, D' is all the *conditional* default value declarations, notated as, $D' = D \backslash D_{unconditional}$.

3. Refined T'

$$T = T_{term} \cup T_{predicate}$$

T_{term} is a set of all *term type declarations*, which assign a type $\tau \in \mathcal{T}$ to a term $t \in \mathbb{HU}$ and is written as $t : \tau$. $T_{predicate}$ is a set of all predicate type declarations. A *predicate type declaration* assigns a type $(\tau_1, ..., \tau_n) \in \mathcal{T}^n$ to predicate $p \in \Pi^n$ and is written as $p : (\tau_1, ..., \tau_n)$, where τ_i is the type of p's i-th argument.

The T_{term} is not related to predicates at all, so it is not taken into account of the similarity between predicates.

Therefore, $T' = T_{predicate}$.

After filtering the RFuzzy program $P = (R, D, T)$ into $P' = (R', D', T')$, the second step is taken to reform D' into D_{new}. A *default value declarations* in D' is

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

Since φ_i are FOL formulas, it always could be in a DNF(Disjunctive Normal Form), $\varphi_i^1 \vee ... \vee \varphi_i^{k_i}$, then $default(p/n) = \delta_i$ if φ_i is written in the *fuzzy clauses* format $A \xleftarrow{c,F_c} F(B_1, ..., B_n)$.

$$p(\vec{x}) \xleftarrow{\delta_i,\cdot} \varphi_i^1$$

$$\vdots$$

$$p(\vec{x}) \xleftarrow{\delta_i,\cdot} \varphi_i^{k_i}$$

Then $default(p/n) = [\delta_1 \text{ if } \varphi_1, ..., \delta_m \text{ if } \varphi_m]$ could be reformed as,

$$p(\vec{x}) \xleftarrow{\delta_1,\cdot} \varphi_1^1$$

$$\vdots$$

$$p(\vec{x}) \xleftarrow{\delta_1,\cdot} \varphi_1^{k_1}$$

$$\vdots$$

$$p(\vec{x}) \xleftarrow{\delta_m,\cdot} \varphi_m^1$$

$$\vdots$$

$$p(\vec{x}) \xleftarrow{\delta_m,\cdot} \varphi_m^{k_m}$$

Apparently, the reforming procedure from D' into D_{new} preserves the semantic equivalence. The result of *reforming* functioning over P' is $P_{new} = (R_{new}, T_{new})$, where $R_{new} = R' \cup D_{new}$ a union of the *fuzzy clauses* and *default value declarations* in the same form of *fuzzy clauses*'s, $T_{new} = T'$.

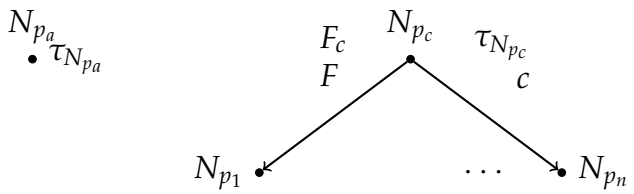


Figure 3.1: Predicate Tree

Constructing the predicate tree

Every predicate in the *fuzzy program* P_{new} could be represented as a tree. A formal description of the approach of generating a corresponding tree for a certain predicate begins with the definition of *atomic predicate* and *complex predicate*.

Definition 3.1.1
(Atomic predicate). Predicates appearing only in the bodies of rules in R_{new}, and never appearing in the heads of any rules, are **atomic predicates**, since they are not defined by any other predicates.

Definition 3.1.2
(Complex predicate). Predicates appearing in the heads of rules in R_{new} are **complex predicates**, in the sense that they could be represented by the bodies of the rules with them as heads.

Atomic predicate is written as p_a, and *complex predicate* as p_c. *Atomic predicate* p_a is represented in a tree-form, which is a node with information τ_{p_a}, and without any children nodes, where τ_{p_a} is the *type* of p_a. *Complex predicate* p_c by definition must appear in the head of some rules in P_{new}, whose form is,

$$p_c(\vec{t}) \xleftarrow{c,F_c} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$

Then the tree of p_c has a root N_{p_c}, with its node information, c, F_c, F, and τ_{p_c}, which is the *type* of p_c. The branches from N_{p_c} are N_{p_1}, ..., N_{p_n}, which are expanded as corresponding trees recursively. Thus, the leaves in the tree are *atomic predicates*, which never appear in the heads of any rules in R_{new}, and can not be expanded any more.

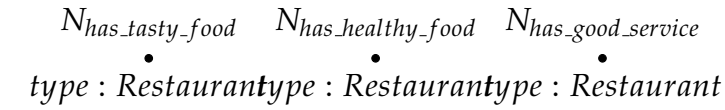
If there are several rules defining a certain predicate p, then there are different corresponding trees for p, since the expanding procedure is recursive, which means, the children of p node could also be defined in several other rules, then more possible trees are generated.

Example 3.1.3
Here is a part of short RFuzzy program,

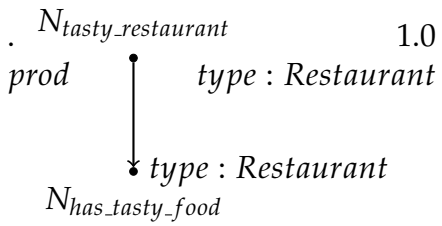
$$\begin{array}{ll} has\_tasty\_food: & (Restaurant) \\ has\_healthy\_food: & (Restaurant) \\ has\_good\_service: & (Restaurant) \\ tasty\_restaurant: & (Restaurant) \\ good\_restaurant: & (Restaurant) \end{array}$$

$$tasty\_restaurant(X) \xleftarrow{1.0,.} prod \quad has\_tasty\_food(X)$$
$$good\_restaurant(X) \xleftarrow{0.8,.} prod \quad has\_healthy\_food(X), has\_good\_service(X)$$
$$Sim(has\_healthy\_food, has\_tasty\_food) = 0.6$$

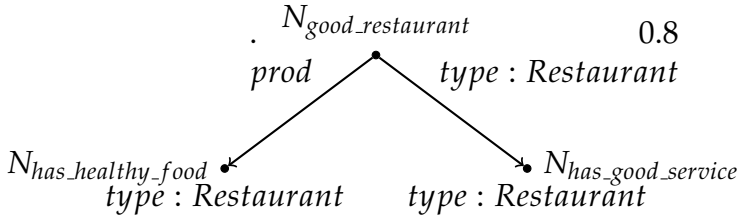
The corresponding trees for atomic predicates *has_tasty_food*, *has_healthy_food*, *has_good_service* are,



The corresponding tree for complex predicate *tasty_restaurant* is,



The corresponding tree for complex predicate *good_restaurant* is,



Equivalence between predicate trees

In order to compare two predicate trees with different number of children, they are reconstructed with the same number of children, preserving the original semantic meaning. For this, the concept "equivalent tree" is introduced in this section.

For each rule in P_{new} that $A \xleftarrow{c,F_c} F(B_1, ..., B_n)$, under its interpretations, it is viewed as a function in the following form,

$$A^{\mathcal{I}} = OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}})$$

where $OP = (\hat{F}_c, \hat{F})$ is a pair, representing the operations over *credit value* c, and other interpretation over atoms B_i in the body of the rule.

Definition 3.1.4
(Identity for complex predicate). There exists a formula α under the operations of RFuzzy Program, which makes

$$OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}}, \alpha^{\mathcal{I}}) = OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}}) \tag{3.1}$$

for each interpretation $\mathcal{I}$. Therefore, the formula α is called *identity* for OP. The existence of α depends on the operation OP.

If *identity* α exists for some OP, then the rule

$$A \xleftarrow{c,F_c} F(B_1, ..., B_n) \tag{3.2}$$

could be rewritten as

$$A \xleftarrow{c,F_c} F(B_1, ..., B_n, \alpha, ..., \alpha) \tag{3.3}$$

Since the procedure of rewriting preserves the semantic equivalence, the corresponding trees of rules 3.2 and 3.3 represent the same semantic meaning.

Definition 3.1.5
(Identity for atomic predicate). For some certain $OP = (\hat{F}_c, \hat{F})$, there exists a formula β under the operations of RFuzzy Program, which makes,

$$A^{\mathcal{I}} = OP(c, A^{\mathcal{I}}, \beta^{\mathcal{I}})$$

for any interpretation $\mathcal{I}$, where c is a real number in the range of $[0, 1]$. Therefore, β is called identity for OP.

Reconstructing predicate trees with equivalence

In this section, the approach of constructing predicates tree with equivalence is displayed, which is used for expanding two comparing nodes with the same structure in the algorithm introduced in section 3.1.2.

- *Atomic predicate*

  Suppose that p_a is a *atomic predicate*. Its type is τ. The identity for some OP associated with p_a is β.

  The corresponding tree of it is a node N_{p_a} with information, which are τ of p_a and a number 0, indicating that p_a is atomic predicate. There are no branches for the node N_{p_a}.

  With the equivalence extension, the corresponding tree is presented in figure 3.2. The root is a node N_{p_a} with information, which are the *type* of p_a τ, the operation OP and atomic mark 0. The branches are a node N'_{p_a} and several nodes N_β. N'_{p_a} carries information τ, which is p_a's type and atomic mark 0. N_β carries information of identity β and identity mark *id*, the number of such nodes depends on two comparing predicates.

- *Complex predicate*

  For *complex predicate*, there would be at least one rule defining it as,

  $$p_c(\vec{t}) \xleftarrow{c,F_c} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$

  and for $OP = (\hat{F}_c, \hat{F})$, there exists an identity α. τ, and τ_i are represented types of p_c and p_i respectively.

  The corresponding tree for p_c is

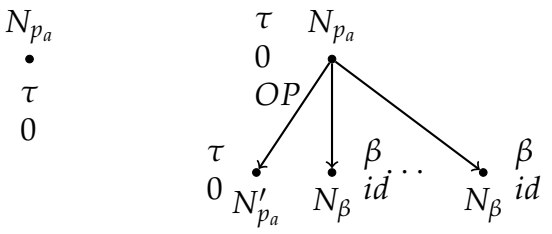
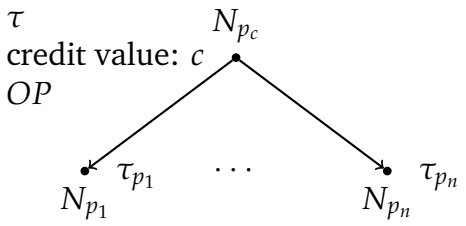
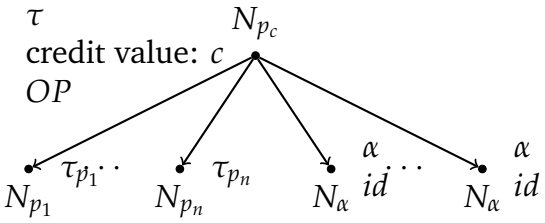


Figure 3.2: Atomic predicate tree with equivalence



where N_{p_c} is the root with information which are τ, OP and credit value c. N_{p_i}s are the branches carrying their *types* as information.

With the equivalence extension, the tree is



N_{p_c} is the root with information which are τ, OP and credit value c. N_{p_i}s are the branches with their *types* as information, and several nodes N_α carrying information of identities α with identity mark id are added as extra branches depending on two comparing predicates.

Example 3.1.6
Continuation of the example 3.1.3, reconstruct predicate trees with equivalence.

$$\begin{array}{ll} has\_tasty\_food: & (Restaurant) \\ has\_healthy\_food: & (Restaurant) \\ has\_good\_service: & (Restaurant) \\ tasty\_restaurant: & (Restaurant) \\ good\_restaurant: & (Restaurant) \end{array}$$

$$\begin{array}{lll} tasty\_restaurant(X) & \xleftarrow{1.0,\cdot} prod & has\_tasty\_food(X) \\ good\_restaurant(X) & \xleftarrow{0.8,\cdot} prod & has\_healthy\_food(X), has\_good\_service(X) \end{array}$$

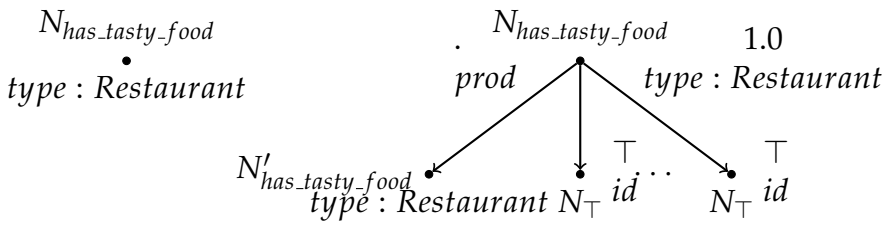


Figure 3.3: two equivalent predicate trees for has_tasty_food

The predicate tree for *has_tasty_food* is represented in figure 3.3.

The predicate tree for *tasty_restaurant* is represented in figure 3.4, and 3.5 which is extended with equivalence.

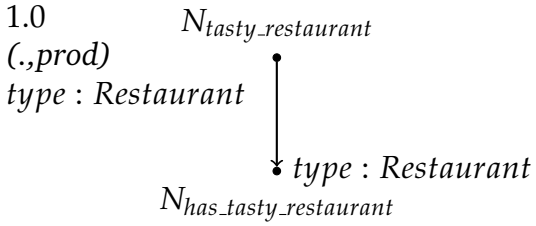


Figure 3.4: predicate tree for tasty_restaurant

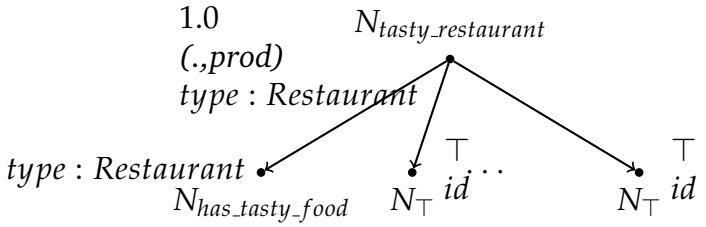


Figure 3.5: predicate tree for tasty_restaurant with equivalence extension

3.1.2 Algorithm for similarity

Let p_a and p_b be arbitrary predicates in RFuzzy Program P. The similarity value is represented as a pair $(level, sim\_degree) \in \mathbb{Z}/\mathbb{Z}^+ \times \{[0,1], u\}$, where *level* is a non-positive integer representing the level of expansion of the corresponding tree, and *sim_degree* is a real number ranging from 0 to 1 representing the similarity degree between two predicates or a mark u to represent that the similarity degree has not been defined.

$(\mathbb{Z}/\mathbb{Z}^+ \times [0,1], \leq)$ is a partial set. Suppose that r_1, r_2 are two real numbers in $[0,1]$, where $r_1 \leq r_2$. We have a partial order such as,

$$\cdots \leq (-3,0) \leq (-3,r_1) \leq (-3,r_2) \leq (-3,1) \leq$$

$$\cdots \leq (0,0) \leq (0,r_1) \leq (0,r_2) \leq (0,1)$$

Therefore, arbitrary two pairs (l_1,v_1), (l_2,v_2) are in $\mathbb{Z}/\mathbb{Z}^+ \times [0,1]$, $(l_1,v_1) \leq (l_2,v_2)$ **iff** $l_1 < l_2$ *or* $l_1 = l_2$ and $v_1 \leq v_2$.

Let p_a and p_b be predicates with types τ_{p_a} and τ_{p_b}, respectively. The similarity between them is defined in the following approach.

- *atomic predicate VS atomic predicate*

  p_a and p_b are *atomic predicates*, which are represented to be isolated nodes with information τ_{p_a} and τ_{p_b} as *type* of p_a and p_b, respectively. If $\tau_{p_a} = \tau_{p_b}$, in the sense that p_a and p_b have the same *type*, then similarity degree between p_a and p_b should be defined directly as a real number in $[0,1]$ in RFuzzy program P as *sim_degree*. Then the similarity value is $Sim(p_a,p_b) = (l, sim\_degree)$, where l is the level of expansion. Once no such information found in RFuzzy program P, return $(-\infty, 0)$ as default similarity value, which means the similarity can not be found even though the predicates could be expanded until infinite level.

Figure 3.6: Comparison between two atomic predicates

- *atomic predicate VS complex predicate*

  p_a is an atomic predicate, and p_b is a complex predicate. Then, p_a is represented as an isolated node with information which are τ_{p_a} and 0 as atomic mark. While, p_b is represented as an isolated node with information τ_{p_b}. The similarity between them is achieved by following,

  1. If $\tau_{p_a} \neq \tau_{p_b}$ then $Sim(p_a, p_b) = (-\infty, 0)$
  2. If $\tau_{p_a} = \tau_{p_b}$, then search for defined *sim_degree* in RFuzzy program, which is a real number in $[0, 1]$. If there exists, then return the similarity value as (l, sim_degree), where l is the level of expansion which p_a and p_b are on.
  3. If $\tau_{p_a} = \tau_{p_b}$ and there doesn't exist the defined similarity in RFuzzy program, then two steps are taken afterwards,
     - Return a middle result $Sim(p_a, p_b) = (l, u)$, where l is the level of expansion which p_a and p_b are on and u represents the similarity degree is undefined.
     - Expand p_a and p_b in the following way. p_b is expanded according to its rule

       $$p_b(\vec{t}) \xleftarrow{c,F_c} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$

       N_{p_b} is the root with information $INFO_{p_b} = \{c, OP = (\hat{F_c}, \hat{F}), \tau_{p_b}\}$. Its branches are nodes N_{p_1}, ..., N_{p_n} with *type* τ_{p_1}, ..., τ_{p_n} of p_1, ..., p_n respectively. p_a is expanded according to OP in $INFO_{p_b}$, then N_{p_a} is the root with $INFO_{p_a} = \{c', OP = (\hat{F_c}, \hat{F}), \tau_{p_a}\}$. Its branches are nodes N'_{p_a}, N_β, ...,N_β. The number of children of N_{p_a} is n, which is the same as N_{p_b}.

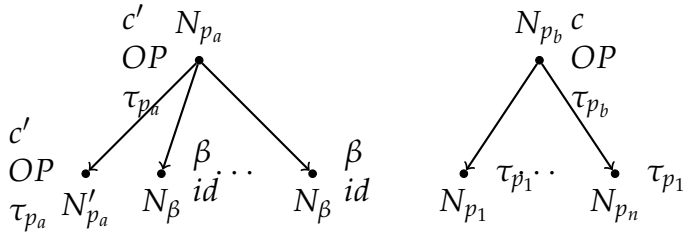


Figure 3.7: Comparison between atomic and complex predicates

- *complex predicate VS complex predicate*

  p_a and p_b are complex predicates with τ_{p_a} and τ_{p_b} as *type*, respectively. If p_a and p_b are of the different *types*, that is, $\tau_{p_a} \neq \tau_{p_b}$, then $Sim(p_a, p_b) = (-\infty, 0)$ will be returned, otherwise, the expanding procedure will be carried on according to the fuzzy rules. p_a is defined by rule

  $$p_a(\vec{t}) \xleftarrow{c^a,F_c^a} F^a(p_1^a(\vec{t_1^a}), ..., p_n^a(\vec{t_n^a}))$$

with $OP_a = (F_c^a, F^a)$ and p_b is defined by rule

$$p_b(\vec{t}) \xleftarrow{c^b, F_c^b} F^b(p_1^b(\vec{t_1^b}), ..., p_m^b(\vec{t_m^b}))$$

with $OP_b = (F_c^b, F^b)$. There are two cases of p_a and p_b expansion.

1. p_a and p_b are defined by the different OP
   If $OP_a \neq OP_b$, which is, $F_c^a \neq F_c^b$ or $F^a \neq F^b$, then return 0 as similarity degree, and $Sim(p_a, p_b) = (-\infty, 0)$.
2. p_a and p_b are defined by the same OP
   If $OP_a = OP_b$, which is $F_c^a = F_c^b$ and $F^a = F^b$, then same two steps are taken here,
   - Return middle result $Sim(p_a, p_b) = (l, u)$
   - Continue the identity formalization, after which, p_a and p_b have corresponding trees with the same construction, where the roots are of the same type, the same OP, and have the same number of children. The comparing procedure of their children is recursively defined.

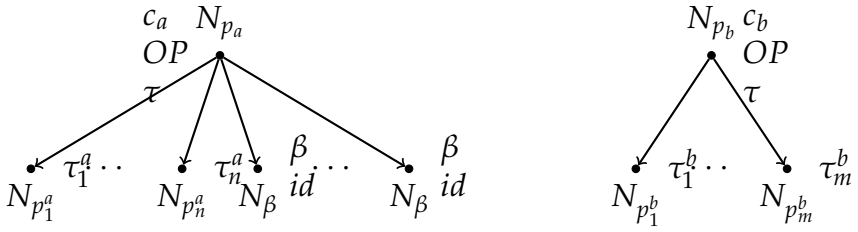


Figure 3.8: Comparison between two complex predicates

For each expansion, the middle result which is the similarity value before next expansion and comparison, must be returned to filter some pairs of corresponding trees to reduce unnecessary expansion or comparison. The middle result of one level is formalized as follow.

Suppose that p_a and p_b are on level l in their own trees, and are satisfied expanding requirements with n children for each. After combination of their children, n optimal $Sim(p_i^a, p_j^b)$ are returned and composed as a *middle set* M_s. The possible values in M_s are $(-\infty, 0)$, $(l-1, sim\_degree)$, $(l-1, u)$, where $sim\_degree \in [0, 1]$. There is one subset of M_s, called *decision middle set*, defined as,

$$M_{ds} = \{(level, degree) | level \neq -\infty, degree \neq u\} \tag{3.4}$$

The *middle result* is $M_r = (l-1, M_v)$, where M_v is defined as,

$$M_v = \frac{\sum_{i=1}^{m} v_i + 1 - |c^a - c^b|}{n+1} \tag{3.5}$$

where m is the cardinality of M_{ds}, v_i is an element in M_{ds}.

The meaning of $1 - |c^a - c^b|$ is as follows:

Credit value c^a and c^b are the trust degrees of fuzzy rules within head p_a and p_b, respectively. They are values in unit interval $[0, 1]$. We define the distance between c^a and c^b by $1 - norm$, which is $|c^a - c^b|$. The similarity between them is $1 - |c^a - c^b|$. c^a and c^b are considered as a special pair of combinations, whose similarity is counted into the middle result of similarity between p_a and p_b.

Furthermore *BMS* requires an extension regarding the identity predicate. Firstly, the purpose of introducing identity α for an operation $OP = (\hat{F}_c, \hat{F})$ is to build two predicate tree in the same structure, where two predicate trees have the identical OP and the same number of children. The procedure of reconstructing predicate tree with equivalence preserves the original semantics. When comparing identity α and an arbitrary predicate p, the similarity between them is defined as $Sim(\alpha, p) = (l, v)$, where l is the level the identity is on, and v is an arbitrary value in $[0, 1]$. The value v will not affect the decision of choosing the optimal combination. The reason is shown in next point.

Lastly, the algorithm is concluded with the termination step. For any comparing pair (p_i^a, p_j^b) from children of p_a and p_b, if $Sim(p_i^a, p_j^b) = (l - 1, u)$, it means that p_i^a and p_j^b have the same type but the similarity degree have not been defined directly in the RFuzzy program, and it could be reached by expanding them according to certain rules. The expanding procedure could be continued until both comparing predicates are atomic, which makes the algorithm terminate.

The algorithm terminates **iff** the comparing trees of predicates are *completely built*.

Definition 3.1.7
Completely Built. The two comparing predicates are represented as trees, and expanded and valued in synchronization with each other until no comparing value $(level, u)$ is returned. Then the comparing trees are *completely built*. The **similarity degree** is calculated from leaves to root by the definition of *middle value*, and the **level** is the lowest level of the comparing trees.

Example 3.1.8
Here is a short RFuzzy program from the example 3.1.3,

$$\begin{array}{ll} has\_tasty\_food: & (Restaurant) \\ has\_healthy\_food: & (Restaurant) \\ has\_good\_service: & (Restaurant) \\ tasty\_restaurant: & (Restaurant) \\ good\_restaurant: & (Restaurant) \end{array}$$

$$\begin{array}{lll} tasty\_restaurant(X) & \xleftarrow{1.0,.} prod & has\_tasty\_food(X) \\ good\_restaurant(X) & \xleftarrow{0.8,.} prod & has\_healthy\_food(X), has\_good\_service(X) \end{array}$$

$$Sim(has\_healthy\_food, has\_tasty\_food) = 0.6$$

The set of *Atomic predicates* is $AP = \{has\_tasty\_food, has\_healthy\_food, has\_good\_service\}$, and the set of *complex predicates* is $CP = \{tasty\_restaurant,$

good_restaurant}. They have the same *type* 'Restaurant'. There are two tasks as examples for gaining the similarity, one is between *tasty_restaurant* and *has_tasty_restaurant*, and the other is between *tasty_restaurant* and *good_restaurant*.

- $Sim(tasty\_restaurant, has\_tasty\_food)$

  Since $tasty\_restaurant \in CP$, and $has\_tasty\_food \in AP$, the procedure of obtaining similarity follows the case "atomic predicate VS complex predicate". In an abbreviation, tr is represented as *tasty_restaurant*, htf is for *has_tasty_food*, R means 'Restaurant'. For this pair of predicates, their *types* are the same, that is,

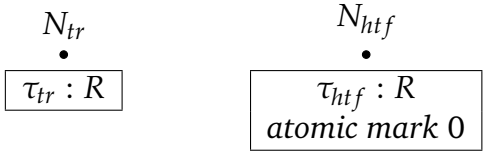


Figure 3.9: Similarity between tr and htf

$\tau_{tr} = \tau_{htf} = R$. And, the similarity value between them is not directly defined in program. Then, two steps are taken afterwards,

1. Return a middle result $Sim_m(tr, htf) = (0, u)$.
2. Expand tr and htf. According to the rule in program, which is

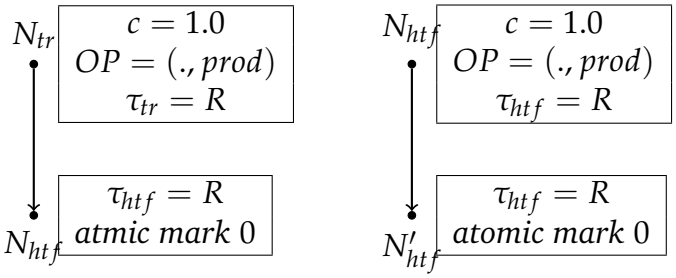


Figure 3.10: Expansion of tr and htf

$$tasty\_restaurant(X) \xleftarrow{1.0,.} prod\ has\_tasty\_food(X)$$

tr is expanded on the left of the figure 3.14, and according to $OP = (., prod)$ from *tr*'s rule, htf is expanded on the right side. As seen in the graph, only one combination of predicates in level -1 can be achieved, which is (htf, htf). The similarity value of them is $(-1, 1)$, since they are the same predicate, the similarity between them is 1.

Thus, the similarity value is 1 between *has_tasty_food* and *tasty_restaurant* achieved in the level -1.

- $Sim(tasty\_restaurant, good\_restaurant)$

  Since $tasty\_restaurant \in CP$, and so is $good\_restaurant$, the procedure of obtaining similarity follows the case "complex predicate VS complex predicate". In an abbreviation, tr is represented as $tasty\_restaurant$, gr is for $good\_restaurant$, R means 'Restaurant'. For this pair of predicates, their *types* are the same, that is,

Figure 3.11: Similarity between tr and gr

$\tau_{tr} = \tau_{gr} = R$. And, the similarity value between them is not directly defined in RFuzzy program. Then, two steps are taken afterwards,

1. Return a middle result $Sim_m(tr, gr) = (0, u)$.
2. Expand tr and gr. gr is expanded acccording to the rule in program, which

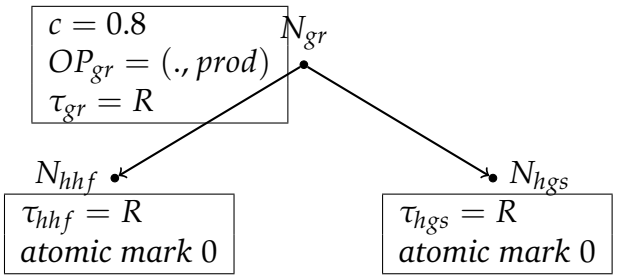


Figure 3.12: Expansion of gr

is

$$good\_restaurant(X) \xleftarrow{0.8,.} prod\ has\_healthy\_food(X), has\_good\_service(X)$$

According to the rule

$$tasty\_restaurant(X) \xleftarrow{1.0,.} prod\ has\_tasty\_food(X)$$

and equivalence extension associated with OP_{gr}, the tr is expanded into, Two combinations of predicates in level -1 can be achieved, which are $M_{s_1} = \{(hhf, htf), (hgs, \top)\}$ and $M_{s_2} = \{(hgs, htf), (hhf, \top)\}$. In our program, only similarity between has_tasty_food and $has\_healthy\_food$ is defined as

$$Sim(has\_healthy\_food, has\_tasty\_food) = 0.6$$

Suppose that for any predicate p, $Sim(\top, p) = 0.5$, then the similarity achieved from M_{s_1} is $M_{r_1} = (-1, 0.6\dot{3})$ and from M_{s_1} is $M_{r_2} = (-1, 0.4\dot{3})$. The former combination M_{s_1} will be chosen. In M_{s_1}, there is no (l, u), which

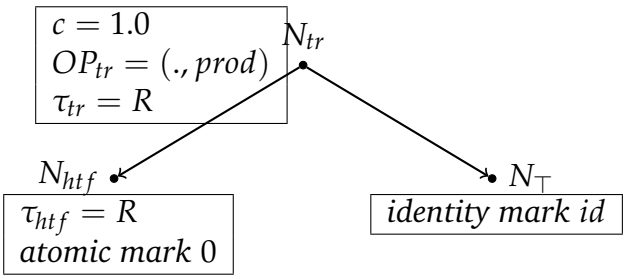


Figure 3.13: Expansion of tr

means no undefined similarity in this level l. Thus the comparing trees are *completely built*, the algorithm terminates, and the similarity value is 0.63 between *good_restaurant* and *tasty_restaurant* achieved in the level -1.

3.2 Shortcomings of *SBM*

Even though the *SBM* methodology is a promising approach, it does not come without some serious shortcomings. These problems lie in three main characteristics of the method, namely:

- Construction of the predicate three
- Search algorithm for similar predicates
- The similarity evaluation function

In this section we demonstrate all these deficiencies with solid examples.

3.2.1 Inclusion of Credibilities in Similarity Evaluation Function

The first defect of the *SBM*approach comes with the way it utilizes the credibility values of the fuzzy rules in the similarity evaluation function for the predicates. The way that *SBM* adopts proves to be problematic when the branching factor is small. Since it's directly summed with the values of similarity pairs in the numerator of the equation, when the cardinality n is small, the values of credibilities simply overshadow of similarity pairs.

Example 3.2.1
The following simple example displays one such scenario:

$$\begin{array}{ll} good\_basketball\_player: & (Player) \\ bad\_basketball\_player: & (Player) \\ good\_technique: & (Player) \\ egoism: & (Player) \end{array}$$

$$\begin{array}{lll} good\_basketball\_player(X) & \overset{0.95,.}{\leftarrow} prod & good\_technique(X). \\ bad\_basketball\_player(X) & \overset{0.9,.}{\leftarrow} prod & egoism(X)\ . \end{array}$$

$$Sim(good\_technique, egoism) = 0.1$$

There is no need for expansion in this case for the predicate trees as they're structurally equivalent. The built trees are depicted in figure 1.14.

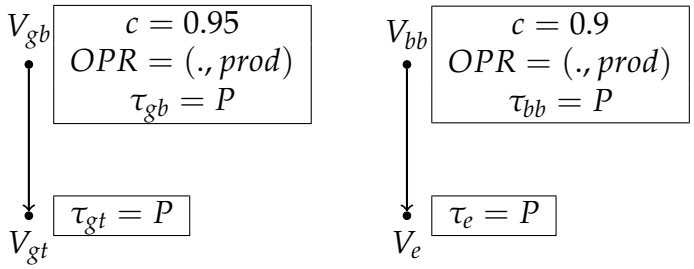


Figure 3.14: Predicate trees for *good_basketball_player* and *bad_basketball_player*

By inspecting the example, one should **not** expect the predicates *good_basketball_player* and *bad_basketball_player* to be similar at all. We shall pursue via performing steps of *SBM* in order to see if the expectation is consistent with the methodology's calculation.

M_s = { (good_technique, egoism)}, $c^a = 0.95$, $c^b = 0.9$, $n = 1$.

$$\begin{aligned} M_v &= \frac{\sum_{i=1}^{1} v_i + 1 - |c^a - c^b|}{n+1} \\ &= \frac{0.1 + 1 - |0.95 - 0.9|}{1+1} \\ &= \frac{1.05}{2} \\ &\cong \mathbf{0.53} \end{aligned} \tag{3.6}$$

So on contrary to what was expected, *SBM* calculates *good_basketball_player* and *bad_basketball_player* to be somewhat similar. As mentioned earlier, this problem is caused by the fact that the equation does not integrate the credibility values in a proper way, thus in some cases (such as the branching factor is low) the emphasize is so high that it dominates the similarity values between the subpredicates.

3.2.2 Convergence to Identity Predicate

Earlier it was stated that in order to compare two predicates, *SBM* needs them to have the same tree structure. And in order to accomplish this when they are not equal, the missing branches and leaves of the smaller tree are filled with identity predicates.

Moreover a default similarity value is defined between an arbitrary predicate and the identity predicate.

Unfortunately this introduces a couple of problems concerning the precision of the algorithm's final result. Similar to the effect that credibility values had in the previous section, this time we may have such an overwhelming impact from the default similarity values that is defined between the identity predicate and an arbitrary predicate. One will especially encounter this kind of scenarios when one tree has high branching factor compared to the other one.

Example 3.2.2
We see an example of such a case in the following program:

$$\begin{array}{ll} classy\_restaurant: & (Restaurant) \\ good\_restaurant: & (Restaurant) \\ well\_trained\_waiters: & (Restaurant) \\ expensive\_inventory: & (Restaurant) \\ has\_good\_service: & (Restaurant) \\ has\_healthy\_food: & (Restaurant) \\ has\_tasty\_food: & (Restaurant) \\ has\_nice\_surroundings: & (Restaurant) \\ has\_high\_reputation: & (Restaurant) \end{array}$$

$$\begin{array}{lll} classy\_restaurant(X) & \overset{1.0,.}{\longleftarrow} prod & well\_trained\_waiters(X), expensive\_inventory(X). \\ good\_restaurant(X) & \overset{0.95,.}{\longleftarrow} prod & has\_healthy\_food(X), has\_good\_service(X), \\ & & has\_nice\_surroundings(X), has\_high\_reputation(X), \\ & & has\_tasty\_food(X). \end{array}$$

$$Sim(well\_trained\_waiters, has\_good\_service) = 0.9$$

$$Sim(expensive\_inventory, has\_nice\_surroundings) = 0.8$$

Regarding to the program we should expect the predicates *classy_restaurant* and *good_restaurant* to have a high similarity degree.

Once again we should start via constructing the predicate trees for *BMS*. As the two trees are not structurally equal, the smaller tree, the predicate tree of *classy_restaurant* must be expanded.

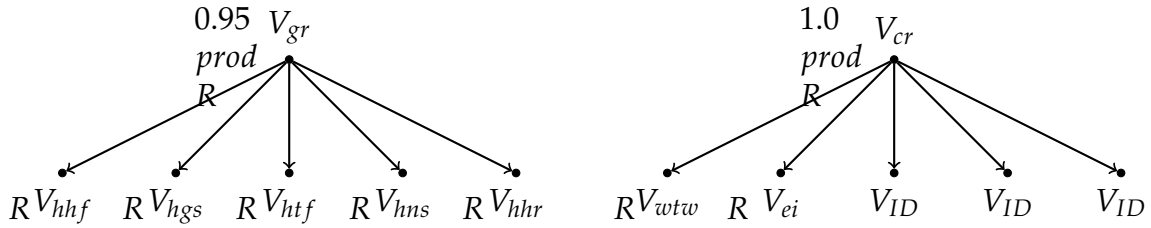


Figure 3.15: Predicate trees in expanded form for *SBM*

There is only one level of expansion. Since at every level the M_s that leads to the highest degree of similarity is chosen, the corresponding decision set will be: M_{ds} = { *(well_trained_waiters, has_good_service), (expensive_inventory, has_nice_surroundings), (ID, has_healthy_food), (ID, has_tasty_food), (ID, has_high_reputation)*}

One thing that we should pay attention to is that, since in this example, *BMS* needed the introduction of the *Identity Predicate*, a default similarity value between an arbitrary predicate and the identity predicate should also be defined. Assume *Sim(ID, p) = 0.3* for any arbitrary predicate.

The rest of the variable values are as follows:

$c^a = 0.95$, $c^b = 1$, $n = 5$.

$$\begin{aligned} M_v &= \frac{\sum_{i=1}^{5} v_i + 1 - |c^a - c^b|}{n+1} \\ &= \frac{2.9 + 1 - |0.95 - 1|}{5+1} \\ &\cong \mathbf{0.55} \end{aligned} \tag{3.7}$$

In this case the algorithm was not able to successfully evaluate and conclude that the predicates are actually closely related. This distortion was caused because of the relatively big branching factor difference and the identity predicates introduced for the missing one. As one can see, in *SBM* as ***the missing number of nodes in one tree increases, the similarity degree calculated by the algorithm converges to the default values*** that is defined between an arbitrary predicate and the identity predicate.

One should suspect that an algorithm which does not introduce any such external knowledge, and just use the original information of the knowledge base, might prevent having such shortcomings.

3.2.3 Wrong Filtering

In her work Lu makes the following assumption [Lu11]:

...However, practically, the number of corresponding trees for certain predicate will not achieve the exponential, since the rules will be defined more reasonable, rather than the given example.

This premise proves to be dangerous as making assumptions on the input setting may cause algorithms to contain flaws.

There is one other faulty behavior of *SBM* which is caused from this relaxed assumption of the the knowledge bases. After every level of expansion, *SBM* checks every predicate pair between the two trees with respect to their resulting similarity degree. Before the next level of expansion is pursued, a filtering is done on the tree by selecting the best pair combination on the level. The problem with this approach is that the information on a prior level is incomplete, and thus wrong steps can be taken when filtering that will cause the loss of crucial information for the main focus, *i.e.* comparing the similarity values of the main predicates.

Since the previous thesis work does not include any examples with predicate trees deeper then just one level, we try to demonstrate the problem with the following example:

Example 3.2.3
The program consists of following type declarations and rules:

$$
\begin{array}{ll}
modern\_city: & (City) \\
livable\_city: & (City) \\
life\_expectancy: & (Society) \\
birth\_rate: & (Society) \\
social\_welfare: & (Society) \\
\#of\_schools & (Society) \\
quality\_of\_academic\_staff & (Society) \\
\#of\_teachers & (Society) \\
compulsory\_schooling\_length & (Society) \\
educated\_society: & (Society) \\
\#of\_healthy\_individuals & (Society) \\
literacy\_rate: & (Society) \\
high\_population: & (Society)
\end{array}
$$

$$
\begin{array}{lll}
livable\_city(X) & \xleftarrow{0.8,.}\ prod & literacy\_rate(X), \#of\_healthy\_individuals(X). \\
literacy\_rate(X) & \xleftarrow{1.0,.}\ prod & compulsory\_schooling\_length(X), \#of\_teachers(X). \\
modern\_city(X) & \xleftarrow{0.7,.}\ prod & educated\_society(X), high\_population(X), \\
 & & social\_welfare(X). \\
educated\_society(X) & \xleftarrow{1.0,.}\ prod & \#of\_schools(X), quality\_of\_academic\_staff(X). \\
high\_population(X) & \xleftarrow{1,0,.}\ prod & birth\_rate(X), life\_expectancy(X).
\end{array}
$$

$$Sim(compulsory\_schooling\_length, \#of\_schools) = 0.9$$

$$Sim(\#of\_teachers, quality\_of\_academic\_staff) = 0.85$$

$$Sim(literacy\_rate, social\_welfare) = 0.4$$

$$Sim(\#of\_healthy\_individuals, life\_expectancy) = 0.7$$

Focus of interest is evaluating the similarity degree of *livable_city* and *modern_city*. Let's start calculating this value via our algorithm.

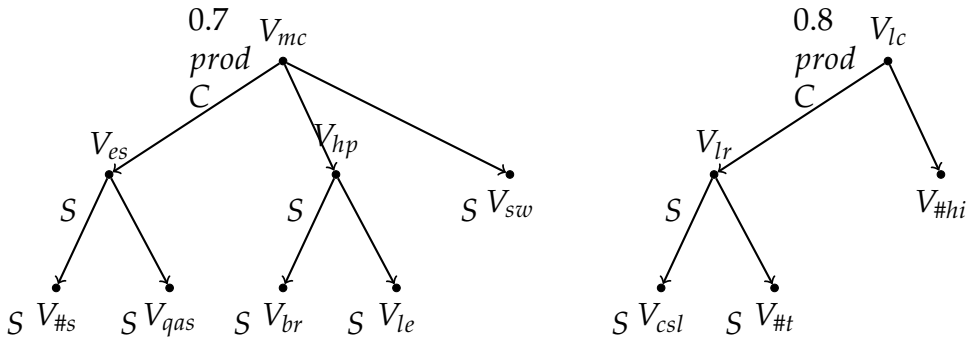


Figure 3.16: Predicate trees of *livable_city* and *modern_city* in original form

By know we know that as trees are do not share the same tree strcuture, *SBM* needs reconstructed versions of the tree before the evaluation algorithm can work.

Figure 3.17: Predicate trees after one level of expansion in *SBM*

After the first expansion is done, again an identity predicate is introduced for the missing leaf node in the predicate tree with the root *livable_city*. The problem arise in this step. As we have discussed before, *SBM* makes a filtering of the node-pairs before the next expansion, with respect to the similarity proximities of the pairs in this level. And once again as we have mentioned, this as most of the information hidden in the lower levels is not apparent to the algorithm yet, filtering can cause neglecting some important paths of the tree.

For instance at this point, between the node pairs, only a similarity relation between the predicates *literacy_rate* and *social_welfare* is defined. The algorithm continues expanding, via selecting the *middle set* with the highest similarity value at the current value as the *decision middle set*. So in this example, at the first level M_{ds} = { (V_{sw} , V_{lr}), (V_{hp} , V_{ID}) } or M_{ds} = { (V_{sw} , V_{lr}), (V_{es} , V_{ID}) } as they prove to be the best of the M_s sets.

However this local optimization approach eliminates other fruitful pair combinations such as *social_welfare* and *educated_society*. In *Figure 6* this pair corresponds to the left subtrees of the main predicate trees. These prove to be the most similar subconcepts of the two as there are two leaf node pairs for which the similarity relation is defined via values *0.9* and *0.85*.

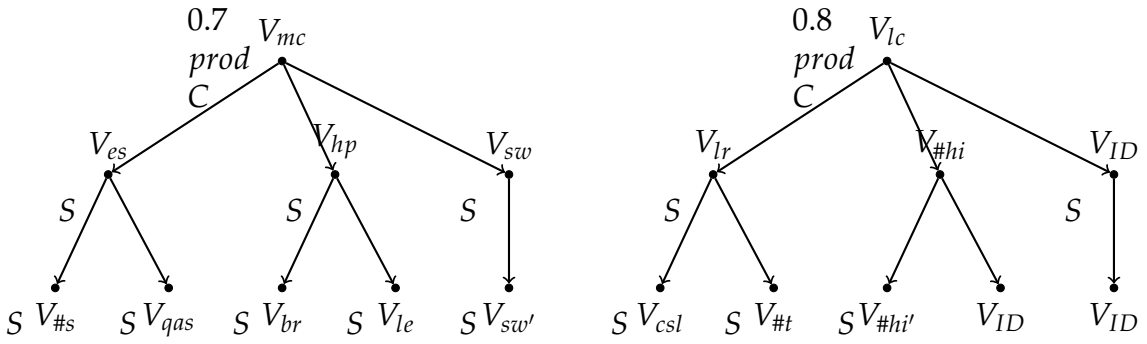


Figure 3.18: Predicate trees after two levels of expansion in *SBM*

All in all, wrong choice for filtering causes comparing wrong pairs of subpredicates in the end and thus a distorted final similarity degree between the predicates of interest.

3.2.4 Shared Similarity Concepts

Another shortcoming of the *SBM* approach is that since it does a one to one mapping between the subconcepts of the main predicates, it can not utilize the information given in the knowledge base completely where a particular concept is included in more then one similarity relations.

Example 3.2.4
Let us observe the following simple program:

$$\begin{array}{ll} touristic\_place: & (Land) \\ nice\_destinationt: & (Land) \\ cultural\_venues: & (Sight) \\ natural\_wonders: & (Sight) \\ many\_sights: & (Sight) \\ good\_weather: & (Temperature) \end{array}$$

$$touristic\_place(X) \xleftarrow{1.0,.}_{prod} cultural\_venues(X), natural\_wonders(X).$$
$$nice\_destination(X) \xleftarrow{1.0,.}_{prod} good\_weather(X), many\_sights(X).$$

$$Sim(cultural\_venues, many\_sights) = 0.7$$

$$Sim(natural\_wonders, many\_sights) = 0.7$$

In the evaluation step, since both of the similarity degrees are equal the algorithm will conclude that there are two optimal M_{ds}. It will either conclude M_{ds} = { (V_{cv} , V_{ms}) } and the pair (V_{nw} , V_{gw}) to be incomparable, or the counter case where M_{ds} = { (V_{nw} , V_{ms}) } and the pair (V_{cv} , V_{gw}) to be incomparable. In both cases the algorithm is bound to miss on the similarity relations and thus the evaluation algorithm can not utilize that data.

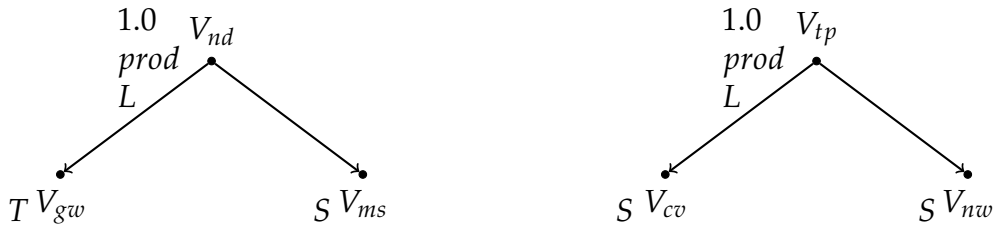


Figure 3.19: Predicate trees for *nice_destination* and *touristic_place*

In the case for which the proximity of similarity of the relations were different, that time the algorithm would simply neglect the smaller one. Thus again half of the information in the knowledge base would be lost without chance of recovery.

CHAPTER 4

THE NEW APPROACH

As it was discussed and depicted, in chapter 3 the main problems of the previous approach are:

- Faulty inclusion of the credibility values for the fuzzy rules, into the similarity evaluation function.
- The final result of the method converging to the default values defined between an arbitrary predicate and the identity predicate.
- Flawed filtering mechanism of the search algorithm.
- No means for utilizing the information which comes from shared similarity concepts in similarity predicate pairings.

Thus hereby we propose a new methodology which in general lines:

- Has a sound evaluation algorithm that prevents credibility values from dominating the final result.
- Does not introduce any extra knowledge which is not originally in the knowledge base, so avoids problems like the final result converging to erroneous values.
- Has a solid search algorithm which is able to utilize all of the information that the knowledge base contain, including similarity concepts that appear in more than one fuzzy rule.
- And as the predicate trees are maintained in their original forms, algorithm is able to work effectively on the complete structure emitting the need of filtering some parts of the data, which could potentially consist of crucial information.

We present the method in the following way:

We lay the fundamentals for a generic mathematical formulation of the main focus of the research. Then the application of the methodology in logic programming is inspected. Lastly, final section sheds light on the comparison of the current and prior approaches via observing solid examples.

4.1 Problem Description

The first step for our approach is generalizing the research's main focus, finding the similarity proximity of two predicates via giving a mathematical foundation. As seen in chapter 3, the solution of the problem relies heavily on the predefined similarity relations between subconcepts. Moreover it was displayed that by expanding the rules in the knowledge bases, the construction of the predicate trees resulted in a graph representation of the original domain. This problem of matching subconcepts in two trees can easily describe the precise meaning of the matching with a mathematical definition. Informally, one may define matching between two predicates in a knowledge base of a logic program as follows: given some specific node corresponding to concept of our interest in the knowledge base- call it a, two concepts (i.e. a and candidate node) can be matched in the corresponding trees if there is a mapping between some specific subset of nodes which are reachable from a in first tree and some subset of nodes which are reachable from candidate node in the second tree.

4.1.1 Preliminaries

A function f is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output.

A *directed graph* (digraph) is an ordered pair *(V, E)* , where V is a set and E is a binary relation on V .

In a digraph *(V, E)*, the elements of V are called vertices, and the elements of E are called the edges of the digraph.

A digraph *(V',E')* is a subgraph of a digraph *(V, E)* if $V' \subseteq V$ and $E' \subseteq E$.

In a digraph (V, E), a path from a vertex u to a vertex u' is a sequence $\langle v_0, v_1, ..., v_k \rangle$ of vertices such that $u = v_0$ and $u' = v_k$,and $(v_{i-1}, v_i) \in E$ for $1 \leq i \leq k$; we call v_k as the terminal node of the path. If there is a path from a vertex u to a vertex v, then we say that v is reachable from u. If V' is a subset of V, a path from u to v whose vertices belong to V' is a path from u to v in V'. If there exists a path from u to v in V' , v is reachable from u in V'.

Let $G_0 = (V_0, E_0)$ be a digraph whose vertices are labeled by a function $f_0 : V_0 \rightarrow VL_0$ where VL_0 is the vertex labels of G_0 and let $G_1 = (V_1, E_1)$ be a graph whose vertices are labeled by a function $f_1 : V_1 \rightarrow VL_1$ where VL_1 is the vertex labels of G_1. Let vm be a function that maps the label of a vertex in VL_0 to a set of vertex labels in VL_1.

We say that a terminal node v_0 matches a terminal node v_1, iff

$$VL_1(v_1) \in vm(VL_0(v_0)).$$

Informally a terminal node v_0 matches a terminal node v_1, when the vertex label of v_1 is in the vertex label set that the vertex label of v_0 is mapped to by vm.

We say that an arbitrary node v_0 matches an arbitrary node v_1, iff

For each element u_0 of a subset of vertices reachable from v_0 via a path p_0, there exists a vertex u_1 reachable from v_1 via a path p_1 in G_1 such that u_0 matches u_1.

Again if we are to state the meaning in an informal manner, two arbitrary nodes v_0 and v_1 match, if and only if there exists a set of vertices that are reachable from v_0, and each of these vertices match to some vertex which are reachable from v_1.

4.1.2 Vertex Matching Problem

We define an Vertex Matching Problem *(VMP)* with the following input:

- Two finite, polynomially bounded directed graphs: $G_0 = (V_0, E_0)$, $G_1 = (V_1, E_1)$
- Vertex Labeling Functions: $f_0 : V_0 \rightarrow VL_0$, $f_1 : V_1 \rightarrow VL_1$
- Vertex Labeling Match Function: $vm : VL_0 \rightarrow 2^{VL_1}$
- Vertices to be matched: $v_0 \in V_0$, $v_1 \in V_1$

4.2 Algorithm in the Domain of Fuzzy Logic

The application of the generic formalism presented in section 4.1 consists of three steps:

- Construction of the predicate tree
- Search algorithm for similar predicates
- The similarity evaluation function

In a nutshell, firstly the predicate trees are constructed via utilizing the fuzzy rules of the program. Then the search algorithm locates similar predicate pairings and collects the corresponding values. Finally the evaluation algorithm computes the resulting similarity proximity value for the predicate pair of interest.

The topics are inspected in detail in the following subsections respectively.

4.2.1 Construction of the Predicate Tree

The process for building the predicate trees follow the same methodology as the one depicted in section 3.1.1. Thus as a brief recap we might state that, for every fuzzy rule in the knowledge base in the following form:

$$p_c(\vec{t}) \xleftarrow{c,F_c} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$

we expand the head of the rule p_c via adding the predicates that the body of the fuzzy rule contain, i.e. $p_1, ..., p_n$, as its children in the directed tree graph. The task continues in a recursive manner until all the rules are exhausted.

No modifications are done on the original predicate trees at any step of the algorithm. Hence the process of building the trees is very straightforward.

4.2.2 Search and Evaluation

Similarity searching process heavily builds on the generic *vertex matching algorithm* given in section 4.1. The only modification is that instead of searching exact matches between the subconcepts, the focus is on finding ones for which a fuzzy similarity rule exists in the particular program.

So if we are to re-state the methodology with the minor modification, when observing the similarity between predicates p_1 and p_2, for every vertex v_1 reachable from p_1 via a path t_1, the algorithm looks for a vertex v_2 reachable from p_2 via a path t_2 where the similarity degree between v_1 and v_2 is defined. The values of these relations are collected in the set *sm* and similarity degree *sd* between p_1 and p_2 is evaluated with the following equation:

$$sd(p_1, p_2) = (1 - |cred_{p_1} - cred_{p_2}|)\mathbf{OP}(\frac{\sum_{i=1}^{n} sm_i}{n}) \tag{4.1}$$

where $cred_1$ and $cred_2$ are the credibility values of the fuzzy rules that contain p_1 and p_2 in their heads respectively, OP is the fuzzy operator that is again defined by those rules, and n is the cardinality of the set *sm*.

Mind that as the focus is on two fuzzy rules at a given time, there might me two distinct OP values for the corresponding rules. In those cases, the generic product operation defined on real number is taken as the default OP value.

4.3 Improvements Over Related Work

The purpose of the section is depicting the contributions the new methodology introduces, specifically the improvements over the related work, the *Structured Based Measurement* that was introduced by Lu [Lu11]. As mentioned in the beginning of this chapter, main sources of advancement are the way the predicate trees are constructed, the methodology for searching the predicate trees, and the structure of the evaluation algorithm. We will re-visit the examples that had been presented in chapter 3 in order to observe how the new methodology compare to *SBM* and also see some extension that it introduces to the domain of interest which gives birth to some interesting features for the framework.

4.3.1 Inclusion of Credibilities in Similarity Evaluation Function

The first difference of the approaches come with the way they utilize the credibility values of the fuzzy rules in the similarity evaluation function for the predicates. As mentioned in the previous chapter, the way that *SBM* adopts proves to be problematic when the branching factor is small. Since it's directly summed with the values of similarity pairs in the numerator of the equation, when the cardinality n is small, the values of credibilities simply overshadow of similarity pairs.

Example 4.3.1
The following simple example displays one such scenario:

$$\begin{array}{ll} good\_basketball\_player: & (Player) \\ bad\_basketball\_player: & (Player) \\ good\_technique: & (Player) \\ egoism: & (Player) \end{array}$$

$$\begin{array}{lll} good\_basketball\_player(X) & \overset{0.95,.}{\longleftarrow} prod & good\_technique(X). \\ bad\_basketball\_player(X) & \overset{0.9,.}{\longleftarrow} prod & egoism(X)\ . \end{array}$$

$$Sim(good\_technique, egoism) = 0.1$$

The predicate trees are the same for both of the approaches since there is no need for expansion in this case as they're structurally equivalent. The built trees are as follows:

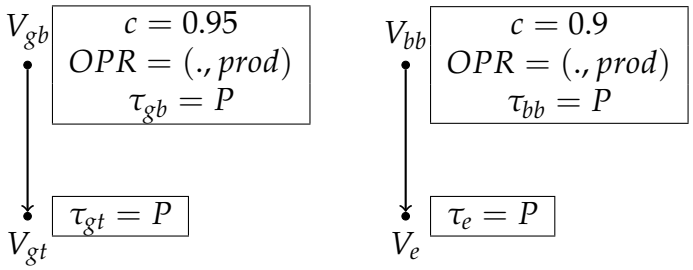


Figure 4.1: Predicate trees for *good_basketball_player* and *bad_basketball_player*

By inspecting the example, one should **not** expect the predicates *good_basketball_player* and *bad_basketball_player* to be similar at all.

This is indeed the case for our approach. Let us evaluate the similarity equation in order to demonstrate this:

In addition to the variables of the equation, we display the similar predicate pairs collected via the search algorithm in the set pm.

$cred_{p_1} = 0.95$, $cred_{p_2} = 0.9$, $OP = prod$, $pm = \{$ (good_technique, egoism)$\}$, $sm = \{0.1\}$, $n = 1$.

When the values are input in the equation:

$$\begin{aligned} sd(p_1, p_2) &= (1 - |cred_{p_1} - cred_{p_2}|)\boldsymbol{OP}(\frac{\sum_{i=1}^{n} sm_i}{n}) \\ &= (1 - |0.95 - 0.9|).(\frac{0,1}{1}) \\ &= 0.095 \\ &\cong \mathbf{0.1} \end{aligned} \tag{4.2}$$

Our method evaluates *good_basketball_player* and *bad_basketball_player* as not similar, which is what we expect by using common sense. In contrast with that, as seen before, *SBM* produces unexpected results. To illustrate this affirmation we show the results *SBM* obtains:

M_s = { (good_technique, egoism)}, $c^a = 0.95$, $c^b = 0.9$, $n = 1$.

$$\begin{aligned} M_v &= \frac{\sum_{i=1}^{1} v_i + 1 - |c^a - c^b|}{n+1} \\ &= \frac{0.1 + 1 - |0.95 - 0.9|}{1+1} \\ &= \frac{1.05}{2} \\ &\cong \mathbf{0.53} \end{aligned} \tag{4.3}$$

So on contrary to what was expected, *SBM* calculated *good_basketball_player* and *bad_basketball_player* to be somewhat similar. As mentioned earlier, this problem is caused by the fact that the equation does not integrate the credibility values in a proper way, thus in some cases(when the branching factor is low) the emphasize is so high that it dominates the similarity values between the subpredicates.

As displayed by the example, our approach handles this issue without any problems as it introduces the confidence values via the operator that is defined by the fuzzy rules themselves.

4.3.2 Convergence to Identity Predicate

We had stated that in order to compare two predicates, *SBM* needs them to have the same tree structure. And in order to accomplish this when they are not equal, the missing branches and leaves of the smaller tree are filled with identity predicates. Moreover a default similarity value is defined between an arbitrary predicate and the identity predicate.

In the same sense that it was discussed earlier, regrettably this introduces a couple of problems concerning the precision of the algorithm's final result. Similar to the effect that credibility values had in the previous section, this time we may have such an overwhelming impact from the default similarity values that is defined between the identity predicate and an arbitrary predicate. We will especially encounter this kind of scenarios when one tree has high branching factor compared to the other one.

Example 4.3.2
Once more we may see an example for that kind of case in the following program:

$$\begin{aligned} &classy\_restaurant: && (Restaurant) \\ &good\_restaurant: && (Restaurant) \\ &well\_trained\_waiters: && (Restaurant) \\ &expensive\_inventory: && (Restaurant) \\ &has\_good\_service: && (Restaurant) \\ &has\_healthy\_food: && (Restaurant) \\ &has\_tasty\_food: && (Restaurant) \\ &has\_nice\_surroundings: && (Restaurant) \\ &has\_high\_reputation: && (Restaurant) \end{aligned}$$

$$
\begin{array}{lll}
classy\_restaurant(X) & \overset{1.0,.}{\longleftarrow} prod & well\_trained\_waiters(X), expensive\_inventory(X). \\
good\_restaurant(X) & \overset{0.95,.}{\longleftarrow} prod & has\_healthy\_food(X), has\_good\_service(X), \\
 & & has\_nice\_surroundings(X), has\_high\_reputation(X), \\
 & & has\_tasty\_food(X).
\end{array}
$$

$$Sim(well\_trained\_waiters, has\_good\_service) = 0.9$$

$$Sim(expensive\_inventory, has\_nice\_surroundings) = 0.8$$

Regarding to the program we should expect the predicates *classy_restaurant* and *good_restaurant* to have a high similarity degree.

Again let's start evaluating the result with our new approach. Our algorithm does not require the predicates to be expanded so the tree are in their original form.

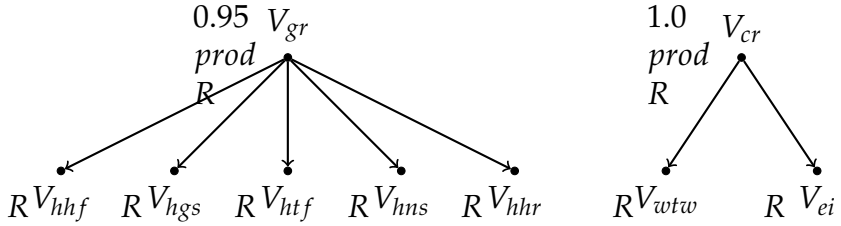


Figure 4.2: Predicate trees in original form

Then the values for the variables of the equation:
$cred_{p_1} = 0.95$, $cred_{p_2} = 1$, $OP = prod$, $pm =$ { (well_trained_waiters, has_good_service), (expensive_inventory, has_nice_surroundings)}, $sm = \{0.9, 0.8\}$, $n = 2$.

And finally the evaluation:

$$
\begin{aligned}
sd(p_1, p_2) &= (1 - |cred_{p_1} - cred_{p_2}|)\boldsymbol{OP}(\frac{\sum_{i=1}^{n} sm_i}{n}) \\
&= (1 - |0.95 - 1|).(\frac{0.9 + 0.8}{2}) \\
&\cong \mathbf{0.81}
\end{aligned}
\tag{4.4}
$$

As the result seems high enough in harmony with the expectation, indeed we can be satisfied with the evaluated degree of similarity.

Thus we may pursue by constructing the predicate trees for *SBM*. As the two trees are not structurally equal, the smaller tree, the predicate tree of *classy_restaurant* must be expanded.

There is just one level of expansion. Since at every level the M_s that leads to the highest degree of similarity is chosen, the corresponding decision set will be: M_{ds} = { *(well_trained_waiters, has_good_service), (expensive_inventory, has_nice_surroundings), (ID, has_healthy_food), (ID, has_tasty_food), (ID, has_high_reputation)*}

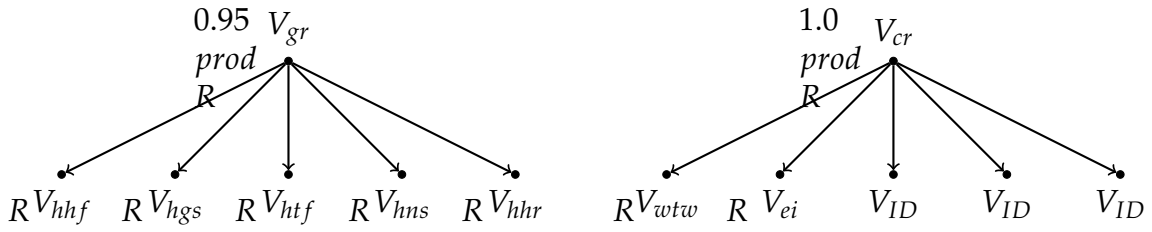


Figure 4.3: Predicate trees in expanded form for *SBM*

One thing that we should pay attention to is that, since in this example, *SBM* needed the introduction of the *Identity Predicate*, a default similarity value between an arbitrary predicate and the identity predicate should also be defined. Assume *Sim(ID, p) = 0.3* for any arbitrary predicate.

The rest of the variable values are as follows:

$c^a = 0.95$, $c^b = 1$, $n = 5$.

$$\begin{aligned} M_v &= \frac{\sum_{i=1}^{5} v_i + 1 - |c^a - c^b|}{n+1} \\ &= \frac{2.9 + 1 - |0.95 - 1|}{5+1} \\ &\cong \mathbf{0.55} \end{aligned} \tag{4.5}$$

In this case the algorithm was not able to successfully evaluate and conclude that the predicates are actually closely related. This distortion was caused because of the relatively big branching factor difference and the identity predicates introduced for the missing one. As one can see, in *SBM* as ***the missing number of nodes in one tree increases, the similarity degree calculated by the algorithm converges to the default values*** that is defined between an arbitrary predicate and the identity predicate.

Since our algorithm does not introduce any such external knowledge, and just use the original information of the knowledge base, it does not have such shortcomings.

4.3.3 Wrong Filtering

As one may remember from the earlier discussions from the previous chapter, there is one other faulty behavior of *SBM* which we suspect to be caused from the relaxed assumption of the knowledge bases. After every level of expansion, *SBM* checks every predicate pair between the two trees with respect to their resulting similarity degree. Before the next level of expansion is pursued, a filtering is done on the tree by selecting the best pair combination on the level. The problem with this approach is that the information on a prior level is incomplete, and thus wrong steps can be taken when filtering that will cause the loss of crucial information for the main focus, *i.e.* comparing the similarity values of the main predicates.

Since the previous thesis work does not include any examples with predicate trees deeper then just one level, we try to demonstrate the problem with the following example:

Example 4.3.3
The program consists of following type declarations and rules:

$$
\begin{array}{ll}
modern\_city: & (City) \\
livable\_city: & (City) \\
life\_expectancy: & (Society) \\
birth\_rate: & (Society) \\
social\_welfare: & (Society) \\
\#of\_schools & (Society) \\
quality\_of\_academic\_staff & (Society) \\
\#of\_teachers & (Society) \\
compulsory\_schooling\_length & (Society) \\
educated\_society: & (Society) \\
\#of\_healthy\_individuals & (Society) \\
literacy\_rate: & (Society) \\
high\_population: & (Society)
\end{array}
$$

$$
\begin{array}{lll}
livable\_city(X) & \xleftarrow{0.8,.} prod & literacy\_rate(X), \#of\_healthy\_individuals(X). \\
literacy\_rate(X) & \xleftarrow{1.0,.} prod & compulsory\_schooling\_length(X), \#of\_teachers(X). \\
modern\_city(X) & \xleftarrow{0.7,.} prod & educated\_society(X), high\_population(X), \\
 & & social\_welfare(X). \\
educated\_society(X) & \xleftarrow{1.0,.} prod & \#of\_schools(X), quality\_of\_academic\_staff(X). \\
high\_population(X) & \xleftarrow{1,0,.} prod & birth\_rate(X), life\_expectancy(X).
\end{array}
$$

$$Sim(compulsory\_schooling\_length, \#of\_schools) = 0.9$$

$$Sim(\#of\_teachers, quality\_of\_academic\_staff) = 0.85$$

$$Sim(literacy\_rate, social\_welfare) = 0.4$$

$$Sim(\#of\_healthy\_individuals, life\_expectancy) = 0.7$$

Focus of interest is evaluating the similarity degree of *livable_city* and *modern_city*. Let's start calculating this value via the new algorithm.

By know we know that our new methodology does not need to modify the trees or do calculations of *mid-results*. Thus let's write down the input needed by the similarity evaluation equation, and then solve the equation for these values.

$cred_{p_1} = 0.7$, $cred_{p_2} = 0.8$, $OP = prod$, pm = { (compulsory_schooling_length, #of_schools), ((#of_teachers, quality_of_academic_staff), (literacy_rate, social_welfare), (#of_healthy_individuals, life_expectancy) }, $sm = \{0.9, 0.85, 0.4, 0.7\}$, $n = 4$.

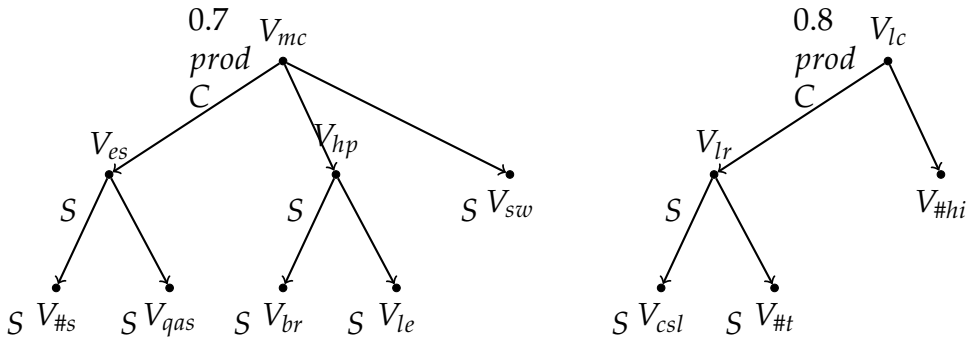


Figure 4.4: Predicate trees of *livable_city* and *modern_city* in original form

$$\begin{aligned} sd(p_1, p_2) &= (1 - |cred_{p_1} - cred_{p_2}|)\boldsymbol{OP}(\frac{\sum_{i=1}^{n} sm_i}{n}) \\ &= (1 - |0.7 - 0.8|).((\frac{\sum_{i=1}^{4} sm_i}{4}) \\ &\cong \mathbf{0.64} \end{aligned} \tag{4.6}$$

Then as usual we follow the steps of *SBM*.

Figure 4.5: Predicate trees after one level of expansion in *SBM*

After the first expansion is done, again an identity predicate is introduced for the missing leaf node in the predicate tree with the root *livable_city*. The problem arise in this step. As we have discussed before, *SBM* makes a filtering of the node-pairs before the next expansion, with respect to the similarity proximities of the pairs in this level. And once again as we have mentioned, this as most of the information hidden in the lower levels is not apparent to the algorithm yet, filtering can cause neglecting some important paths of the tree.

For instance at this point, between the node pairs, only a similarity relation between the predicates *literacy_rate* and *social_welfare* is defined. The algorithm continues expanding, via selecting the *middle set* with the highest similarity value at the current value as the *decision middle set*. So in this example, at the first level M_{ds} = { (V_{sw} , V_{lr}), (V_{hp} , V_{ID}) } or M_{ds} = { (V_{sw} , V_{lr}), (V_{es} , V_{ID}) } as they prove to be the best of the M_s sets.

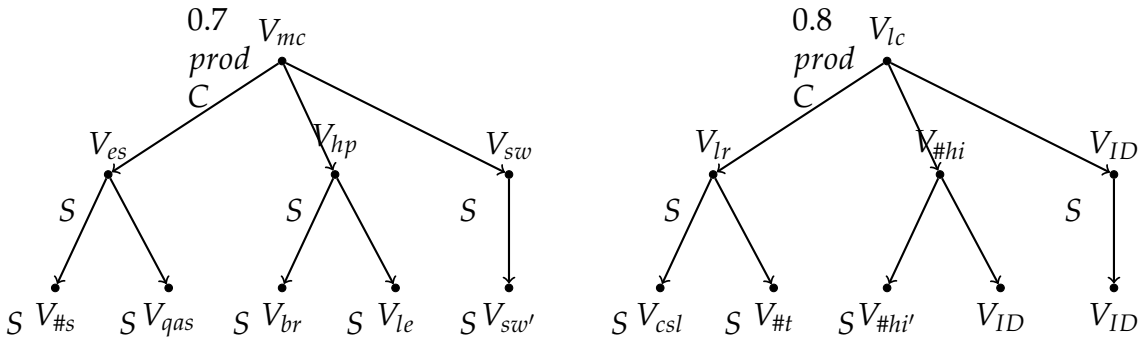


Figure 4.6: Predicate trees after two levels of expansion in *SBM*

However this local optimization approach eliminates other fruitful pair combinations such as *social_welfare* and *educated_society*. In *Figure 6* this pair corresponds to the left subtrees of the main predicate trees. These prove to be the most similar subconcepts of the two as there are two leaf node pairs for which the similarity relation is defined via values *0.9* and *0.85*.

All in all, wrong choice for filtering causes comparing wrong pairs of subpredicates in the end and thus a distorted final similarity degree between the predicates of interest.

4.3.4 Shared Similarity Concepts

In section 3.2.4, we had discussed the incapability of the *SBM* realizing the information where a specific concept is included in more then one similarity relations as the methodology seeks for a one to one mapping between the subconcepts of the main predicates. Our algorithm does not suffer from the same problem as it may collect and process all distinct information from each different fuzzy similarity rules.

Example 4.3.4
Let us once again observe the following simple program:

$$\begin{aligned} &touristic\_place: && (Land)\\ &nice\_destinationt: && (Land)\\ &cultural\_venues: && (Sight)\\ &natural\_wonders: && (Sight)\\ &many\_sights: && (Sight)\\ &good\_weather: && (Temperature) \end{aligned}$$

$$touristic\_place(X) \xleftarrow{1.0,.} prod \quad cultural\_venues(X), natural\_wonders(X).$$
$$nice\_destination(X) \xleftarrow{1.0,.} prod \quad good\_weather(X), many\_sights(X).$$

$$Sim(cultural\_venues, many\_sights) = 0.7$$

$$Sim(natural\_wonders, many\_sights) = 0.7$$

In section 3.2.4 we had proved that the algorithm is bound to miss one of the similarity relations and thus the corresponding evaluation algorithm could not utilize the complete data.

Let us tackle the problem with our methodology:

$cred_{p_1} = 1$, $cred_{p_2} = 1$, $OP = prod$, $pm = \{$ (cultural_venues, many_sights), (natural_wonders, many_sights)$\}$, $sm = \{0.7, 0.7\}$, $n = 2$.

And finally the evaluation:

$$\begin{aligned} sd(p_1, p_2) &= (1 - |cred_{p_1} - cred_{p_2}|)\mathbf{OP}(\frac{\sum_{i=1}^{n} sm_i}{n}) \\ &= (1 - |1 - 1|).(\frac{0.7 + 0.7}{2}) \\ &\cong \mathbf{0.7} \end{aligned} \tag{4.7}$$

As foreseen, the search algorithm were able to realize both of the similarity relations and hence the evaluation algorithm had concluded the accurate similarity degree, which is 0.7 for the case.

Figure 4.7: Predicate trees for *nice_destination* and *touristic_place*

4.3.5 Credibility of Similarity Relations

The domain of interest can be enriched by computing not only the similarity value between predicates, but also the credibility value this similarity has via the introduction of the *fuzzy similarity predicates* with *fuzzy credibility values*.

In the previous research all fuzzy similarity relations are defined without credibility values as if we assume that the similarity computed was always the expected one. So the following type of notation,

$$Sim(good\_technique, swift\_shot) = 0.7$$

can be replaced by:

$$Sim(good\_technique, swift\_shot) \quad \xleftarrow{1.0} 0.7.$$

This extension could be handled by our algorithm with just a small modification in the evaluation equation. In the part where the similarity degree of the collected pairs'

are summed, we should normalize each of these values by the rule's credibility. Thus the algorithm is re-written as:

$$sd(p_1, p_2) = (1 - |cred_{p_1} - cred_{p_2}|)\mathbf{OP}(\frac{\sum_{i=1}^{n}(sm_i \times w_i)}{n}) \tag{4.8}$$

Example 4.3.5
Let's revisit Ex. 2.3., but this time suppose we are sure of the similarity relations with credibility values of 0.7 and 0.6 respectively.

$$\begin{array}{ll} classy\_restaurant: & (Restaurant) \\ good\_restaurant: & (Restaurant) \\ well\_trained\_waiters: & (Restaurant) \\ expensive\_inventory: & (Restaurant) \\ has\_good\_service: & (Restaurant) \\ has\_healthy\_food: & (Restaurant) \\ has\_tasty\_food: & (Restaurant) \\ has\_nice\_surroundings: & (Restaurant) \\ has\_high\_reputation: & (Restaurant) \end{array}$$

$$\begin{array}{lll} classy\_restaurant(X) & \overset{1.0,.}{\longleftarrow} prod & well\_trained\_waiters(X), expensive\_inventory(X). \\ good\_restaurant(X) & \overset{0.95,.}{\longleftarrow} prod & has\_healthy\_food(X), has\_good\_service(X), has\_tasty\_food(X) \\ & & has\_nice\_surroundings(X), has\_high\_reputation(X). \end{array}$$

$$\begin{array}{ll} Sim(well\_trained\_waiters, has\_good\_service) & \overset{0.7}{\longleftarrow} 0.9. \\ Sim(expensive\_inventory, has\_nice\_surroundings) & \overset{0.6}{\longleftarrow} 0.8. \end{array}$$

The only change in our search procedure is that, now the collected similarity values are stored together with their associated credibility value.

Thus the new result from the modified evaluation algorithms follows as:

$$\begin{aligned} sd(p_1, p_2) &= (1 - |cred_{p_1} - cred_{p_2}|)\mathbf{OP}(\frac{\sum_{i=1}^{n}(sm_i \times w_i)}{n}) \\ &= (1 - |0.95 - 1|).(\frac{(0.9 \times 0.7) + (0.8 \times 0.6)}{2}) \\ &\cong \mathbf{0.53} \end{aligned} \tag{4.9}$$

Note that the evaluated similarity value decreased to *0.53* from it's original value of *0.81*. This was expected as the similarity relations' credibility decreased to *0.7* and *0.6* from *1.0*, since they used to be treated as if they were facts.

CHAPTER 5

THE CREDIBILITY VALUE OF THE SIMILARITY RELATIONS

In chapter 3 the *SBM* approach[Lu11] itself and its shortcomings were discussed in detail. Our new methodology was introduced in chapter 4 which overcomes the flows of the preceding approach via the path it adopts for representing the knowledge base, the algorithm it embraces in order to search the data, and the structured function it contains for evaluating the result.

With all its promising traits, the new methodology is not without flaws. Mind that when calculating the final degree of similarity between two predicates, solely the predefined similarity relations on subconcepts are taken into account in the evaluation algorithm. However in real world applications, knowledge bases do not always come with complete information. Thus a notion that emphasizes the approximate error in the accuracy of the result of the algorithm should be introduced, for the sake of the highly probable possibility of encountering missing information.

Example 5.0.6
Consider the following example:

$$
\begin{array}{ll}
decent\_child: & (Person) \\
eats\_well: & (Feature) \\
does\_sports: & (Feature) \\
tidies\_room: & (Feature) \\
studies\_hard: & (Feature) \\
respects\_elders: & (Feature) \\
role\_model\_youngster: & (Person) \\
good\_dietary: & (Feature) \\
fit: & (Feature) \\
neat: & (Feature) \\
good\_grades: & (Feature) \\
obedient\_toParents: & (Feature)
\end{array}
$$

$$
\begin{array}{lll}
decent\_child(X) & \overset{1.0,.}{\longleftarrow} prod & eats\_well(X), does\_sports(X), tidies\_room(X), \\
 & & studies\_hard(X), respects\_elders(X). \\
role\_model\_youngster(X) & \overset{1.0,.}{\longleftarrow} prod & good\_dietary(X), fit(X), neat(X), \\
 & & good\_grades(X), obedient\_toParents(X).
\end{array}
$$

$$Sim(tidies\_room, neat) = 0.8$$

$$Sim(respects\_elders, obedient\_toParents) = 0.9$$

When the proximity of similarity between the predicates *decent_child* and *role_model_youngster* is queried, our algorithm take into consideration only the two predefined similarity relations between the subconcepts. Namely, the ones between *tidies_room* and *neat*, and *respects_elders* and *obedient_toParents*.

However if one reasons on the real-world semantic meanings of the predicates, s/he will realize that the subconcepts pairs

- *eats_well, good_dietary*
- *studies_hard, good_grades*
- *does_sports, fit*

seem to be immensely related, or in other terms *similar* to each other. So indeed one in this case suspect the knowledge base to be missing information. With this motivation, we propose three possible models for evaluating a credibility approximation for our algorithm in the following sections. Namely:

1. Vertex Approach
2. Edge Approach
3. Hybrid Approach

In section 4.3.5 we have seen that our method has the means of representing similarity relations with fuzzy credibility values. With that idea in mind, here we will propose the following:

Assume that for two predicates p_0 and p_1, our algorithm concludes the similarity value S with the corresponding credibility value C. Then the following fuzzy rule is added into the knowledge base when our method terminates.

$Sim(p_0, p_1) \quad \overset{C}{\longleftarrow} S.$

5.0.6 Vertex Approach

The most naive method for computing an estimate of credibility for our problem seems to be finding the percentage of the subconcepts which appear in a similarity relation. The equation is as follows:

$$Credibility = \frac{\#\_of\_subconcepts\_participating\_in\_a\_similarity\_relation}{\#\_of\_total\_subconcepts} \tag{5.1}$$

In the previous example there were five subconcepts for each of the predicates, and two from each side were included in a similarity relation. Hence the degree of credibility of our algorithm for the case of example 5.0.6 is going to be:

$$\begin{aligned} Credibility &= \frac{\#\_of\_subconcepts\_participating\_in\_a\_similarity\_relation}{\#\_of\_total\_subconcepts} \\ &= \frac{4}{10} \\ &= \mathbf{0.4} \end{aligned} \tag{5.2}$$

Then this means the credibility value that we attain to our result is *0.4* for this example. The evaluation function would conclude the similarity degree between the predicates *decent_child* and *role_model_youngster* as *0.85* hence in the end the following fuzzy rule would be added into the knowledge base:

$Sim(decent\_child, role\_model\_youngster) \xleftarrow{0.4} 0.85.$

5.0.7 Edge Approach

There seems to be one point of concern with the approach in section 5.0.6. We want to kindly remind the reader that in section 4.3.4 we had shown that our algorithm was fully able to handle the cases where a specific combination of concepts are included in more than one similarity relations.

Example 5.0.7
Suppose that we extend the example 5.0.6 with the following two similarity relations:

$$Sim(tidies\_room, obedient\_toParents) = 0.6$$

$$Sim(respects\_elders, neat) = 0.55$$

Considering the existing similarity relations, now we have four in total. Another important point that demands attention is that four subconcepts have been included in more than one fuzzy similarity relations. All in all at this points we have more information with respect to the first version of the problem. Thus we would expect the credibility that we attain to our estimation to be bigger. Let us do the calculation with section 5.0.6 approach once again:

$$\begin{aligned} Credibility &= \frac{\#\_of\_subconcepts\_participating\_in\_a\_similarity\_relation}{\#\_of\_total\_subconcepts} \\ &= \frac{4}{10} \\ &= \mathbf{0.4} \end{aligned} \tag{5.3}$$

As can be seen from the results, nothing has changed. That is because the prior approach only considers the number of predicates that are contained in similarity

rules, or vertices in graph terms. It does not able to realize the information of the additional info introduced by the proximity of similarity between different pairings of these nodes, or speaking in graph terms, number of added edges between the two trees. Thus here we propose a second equation which would handle this shortcoming.

$$Credibility = \frac{\#\_of\_similarity\_pairings\_between\_subconcepts}{\#\_of\_possible\_similarity\_pairings\_between\_subconcepts} \tag{5.4}$$

Let us compute the results of both versions of the problem via utilizing this approach respectively.

$$\begin{aligned} Credibility &= \frac{\#\_of\_similarity\_pairings\_between\_subconcepts}{\#\_of\_possible\_similarity\_pairings\_between\_subconcepts} \\ &= \frac{2}{5 \times 5} \\ &= \mathbf{0.08} \end{aligned} \tag{5.5}$$

$$\begin{aligned} Credibility &= \frac{\#\_of\_similarity\_pairings\_between\_subconcepts}{\#\_of\_possible\_similarity\_pairings\_between\_subconcepts} \\ &= \frac{4}{5 \times 5} \\ &= \mathbf{0.16} \end{aligned} \tag{5.6}$$

As expected the second result proved to be bigger, which means the approach was able to attain a higher credibility value for the case where the knowledge base consisted more information.

The only problem with this approach appears to be the big decrease in all of the credibility values compared to the first approach. Here we are dividing the number of existing similarity pairings, to the number of all possible pairings which actually proves to be a relatively huge number. And indeed in reality we wouldn't normally expect for every subconcept to be related, or similar to every other subconcept from the second predicate tree. Thus it seems that this equation seems to go under some sort of normalization process in order to depict more accurate results. This is the motivation for the last approach which is described in section 5.0.8.

5.0.8 Hybrid Approach

As a quick recap, let us see what are the advantages and disadvantages of the approaches presented in sections 5.0.6 and 5.0.7.

	PROS	CONS
Vertex Approach	Realistic results	No means to differentiate shared concepts
Edge Approach	Utilizes complete info	Unrealistic results

Vertex Approach evaluates results that are in harmony with the common sense. However it can not realize the extra knowledge gained by the shared concepts. On the other hand, *Edge Approach* is able to tackle this problem, whereas the credibility values it returns are dramatically lower than the expected ones.

Somehow we would like to come up with an algorithm which would have both approaches' strong points, meanwhile it should avoid the shortcomings of the two. In the light of this motivation, we propose a hybrid approach which merges two approaches together by attaining them some weights:

$$\textit{Credibility} = (w \times \textbf{[Vertex\_Approach]}) + ((1-w) \times \textbf{[Edge\_Approach]}) \tag{5.7}$$

where $w \in [0..1]$.

The intuition tells to select w relatively high (*i.e. bigger then 0.5*) as the first approach was the one with more realistic results, but we wanted to integrate the second approach as we wanted to make use of all the info that the knowledge base contain.

Hence with the assumption of $w = 0.8$, let us calculate both versions of the problem with the latest approach:

$$\begin{aligned} \textit{Credibility} &= (w \times \textbf{[Vertex\_Approach]}) + ((1-w) \times \textbf{[Edge\_Approach]}) \\ &= (0.8 \times \frac{4}{10}) + (0.2 \times \frac{2}{5 \times 5}) \\ &= \mathbf{0.336} \end{aligned} \tag{5.8}$$

$$\begin{aligned} \textit{Credibility} &= (w \times \textbf{[Vertex\_Approach]}) + ((1-w) \times \textbf{[Edge\_Approach]}) \\ &= (0.8 \times \frac{4}{10}) + (0.2 \times \frac{4}{5 \times 5}) \\ &= \mathbf{0.352} \end{aligned} \tag{5.9}$$

Even though it's small, we are able to differentiate the effect of the added knowledge from the distinct results. Moreover final values seem to be meaningful, on contrary of the case in section 5.0.7.

All in all, with the help of our advanced similarity and credibility approximation algorithms, confidently we would be able to add the following fuzzy rules into the corresponding knowledge bases.

$Sim(decent\_child, role\_model\_youngster) \overset{0.336}{\longleftarrow} 0.85.$

$Sim(decent\_child, role\_model\_youngster) \overset{0.352}{\longleftarrow} 0.713.$

5.0.9 Current Directions

As seen in the preceding section, we adopt an intuitionistic way for defining the weights in the credibility evaluation equation of the *Hybrid Approach*. This is a naive first look to an intriguing potential area of research.

In *RFuzzy* framework [MHPCS10], they utilize Multi Adjoint-Logic in order to compute the credibility values from real-world information in an automized fashion. The focus of our ongoing work is centered on this promising idea of developing an atomization method for attaining weights to the evaluation function, thus computing credibility values directly from the examples of real-world data. With this inspiration, in the following chapter we introduce our automized methodology for the concerning problem, illustrating it on a real-world case. There exist many more approaches in other scientific fields that we could observe as a source of inspiration, such as the methods utilized in artificial neural networks.

CHAPTER 6

PRACTICAL APPLICATION

In chapter 3 we introduce our own methodology for evaluating the similarity degree of fuzzy predicates. The method lacked a feature for realizing an error tolerance when working on knowledge bases with incomplete information. For that reason, three approaches are proposed in chapter 5. The most elegant of these approaches, namely the *Hybrid Approach* inherits the best features of the preceding ones, while avoiding the shortcomings via merging the two by attaining them some weights. Previously we displayed how these weights could be decided in an intuitionistic way, and hinted the possibility of an atomized procedure. In this chapter we show our proposal for atomizing the process with a real-world example.

6.1 The Problem

In section 1 we mention about how today's search engines have limited querying capabilities. We give the example of the query *red car*, and say that an ideal frame work would return us the results of the query *orange car* in addition to the original one, with lower credibility values. In the light of this, as our real-world practical example, we inspect the similarity relations between the colors. We may observe colors as a domain of interest in many distinct fields, however one particular interesting example is the set of canonical colors that the web browsers use, namely the *X11 colors* [Wri08]. The system makes use of the *RGB* framework, where colors are specified as triplets. Every color is depicted by three values which represent the *Red*, *Green* and *Blue* amount that the color consists of. The complete *X11* table is shown in the following page.

HTML name	Hex code R G B	Decimal code R G B
Red colors		
IndianRed	CD 5C 5C	205 92 92
LightCoral	F0 80 80	240 128 128
Salmon	FA 80 72	250 128 114
DarkSalmon	E9 96 7A	233 150 122
LightSalmon	FF A0 7A	255 160 122
Red	FF 00 00	255 0 0
Crimson	DC 14 3C	220 20 60
FireBrick	B2 22 22	178 34 34
DarkRed	8B 00 00	139 0 0
Pink colors		
Pink	FF C0 CB	255 192 203
LightPink	FF B6 C1	255 182 193
HotPink	FF 69 B4	255 105 180
DeepPink	FF 14 93	255 20 147
MediumVioletRed	C7 15 85	199 21 133
PaleVioletRed	DB 70 93	219 112 147
Orange colors		
LightSalmon	FF A0 7A	255 160 122
Coral	FF 7F 50	255 127 80
Tomato	FF 63 47	255 99 71
OrangeRed	FF 45 00	255 69 0
DarkOrange	FF 8C 00	255 140 0
Orange	FF A5 00	255 165 0
Yellow colors		
Gold	FF D7 00	255 215 0
Yellow	FF FF 00	255 255 0
LightYellow	FF FF E0	255 255 224
LemonChiffon	FF FA CD	255 250 205
LightGoldenrodYellow	FA FA D2	250 250 210
PapayaWhip	FF EF D5	255 239 213
Moccasin	FF E4 B5	255 228 181
PeachPuff	FF DA B9	255 218 185
PaleGoldenrod	EE E8 AA	238 232 170
Khaki	F0 E6 8C	240 230 140
DarkKhaki	BD B7 6B	189 183 107
Purple colors		
Lavender	E6 E6 FA	230 230 250
Thistle	D8 BF D8	216 191 216
Plum	DD A0 DD	221 160 221
Violet	EE 82 EE	238 130 238
Orchid	DA 70 D6	218 112 214
Fuchsia	FF 00 FF	255 0 255
Magenta	FF 00 FF	255 0 255
MediumOrchid	BA 55 D3	186 85 211
MediumPurple	93 70 DB	147 112 219
BlueViolet	8A 2B E2	138 43 226
DarkViolet	94 00 D3	148 0 211
DarkOrchid	99 32 CC	153 50 204
DarkMagenta	8B 00 8B	139 0 139
Purple	80 00 80	128 0 128
Indigo	4B 00 82	75 0 130
DarkSlateBlue	48 3D 8B	72 61 139

HTML name	Hex code R G B	Decimal code R G B
Green colors		
GreenYellow	AD FF 2F	173 255 47
Chartreuse	7F FF 00	127 255 0
LawnGreen	7C FC 00	124 252 0
Lime	00 FF 00	0 255 0
LimeGreen	32 CD 32	50 205 50
PaleGreen	98 FB 98	152 251 152
LightGreen	90 EE 90	144 238 144
MediumSpringGreen	00 FA 9A	0 250 154
SpringGreen	00 FF 7F	0 255 127
MediumSeaGreen	3C B3 71	60 179 113
SeaGreen	2E 8B 57	46 139 87
ForestGreen	22 8B 22	34 139 34
Green	00 80 00	0 128 0
DarkGreen	00 64 00	0 100 0
YellowGreen	9A CD 32	154 205 50
OliveDrab	6B 8E 23	107 142 35
Olive	80 80 00	128 128 0
DarkOliveGreen	55 6B 2F	85 107 47
MediumAquamarine	66 CD AA	102 205 170
DarkSeaGreen	8F BC 8F	143 188 143
LightSeaGreen	20 B2 AA	32 178 170
DarkCyan	00 8B 8B	0 139 139
Teal	00 80 80	0 128 128
Blue/Cyan colors		
Aqua	00 FF FF	0 255 255
Cyan	00 FF FF	0 255 255
LightCyan	E0 FF FF	224 255 255
PaleTurquoise	AF EE EE	175 238 238
Aquamarine	7F FF D4	127 255 212
Turquoise	40 E0 D0	64 224 208
MediumTurquoise	48 D1 CC	72 209 204
DarkTurquoise	00 CE D1	0 206 209
CadetBlue	5F 9E A0	95 158 160
SteelBlue	46 82 B4	70 130 180
LightSteelBlue	B0 C4 DE	176 196 222
PowderBlue	B0 E0 E6	176 224 230
LightBlue	AD D8 E6	173 216 230
SkyBlue	87 CE EB	135 206 235
LightSkyBlue	87 CE FA	135 206 250
DeepSkyBlue	00 BF FF	0 191 255
DodgerBlue	1E 90 FF	30 144 255
CornflowerBlue	64 95 ED	100 149 237
RoyalBlue	41 69 E1	65 105 225
Blue	00 00 FF	0 0 255
MediumBlue	00 00 CD	0 0 205
DarkBlue	00 00 8B	0 0 139
Navy	00 00 80	0 0 128
MidnightBlue	19 19 70	25 25 112

HTML name	Hex code R G B	Decimal code R G B
Brown colors		
Cornsilk	FF F8 DC	255 248 220
BlanchedAlmond	FF EB CD	255 235 205
Bisque	FF E4 C4	255 228 196
NavajoWhite	FF DE AD	255 222 173
Wheat	F5 DE B3	245 222 179
BurlyWood	DE B8 87	222 184 135
Tan	D2 B4 8C	210 180 140
RosyBrown	BC 8F 8F	188 143 143
SandyBrown	F4 A4 60	244 164 96
Goldenrod	DA A5 20	218 165 32
DarkGoldenrod	B8 86 0B	184 134 11
Peru	CD 85 3F	205 133 63
Chocolate	D2 69 1E	210 105 30
SaddleBrown	8B 45 13	139 69 19
Sienna	A0 52 2D	160 82 45
Brown	A5 2A 2A	165 42 42
Maroon	80 00 00	128 0 0
White colors		
White	FF FF FF	255 255 255
Snow	FF FA FA	255 250 250
Honeydew	F0 FF F0	240 255 240
MintCream	F5 FF FA	245 255 250
Azure	F0 FF FF	240 255 255
AliceBlue	F0 F8 FF	240 248 255
GhostWhite	F8 F8 FF	248 248 255
WhiteSmoke	F5 F5 F5	245 245 245
Seashell	FF F5 EE	255 245 238
Beige	F5 F5 DC	245 245 220
OldLace	FD F5 E6	253 245 230
FloralWhite	FF FA F0	255 250 240
Ivory	FF FF F0	255 255 240
AntiqueWhite	FA EB D7	250 235 215
Linen	FA F0 E6	250 240 230
LavenderBlush	FF F0 F5	255 240 245
MistyRose	FF E4 E1	255 228 225
Gray colors		
Gainsboro	DC DC DC	220 220 220
LightGrey	D3 D3 D3	211 211 211
Silver	C0 C0 C0	192 192 192
DarkGray	A9 A9 A9	169 169 169
Gray	80 80 80	128 128 128
DimGray	69 69 69	105 105 105
LightSlateGray	77 88 99	119 136 153
SlateGray	70 80 90	112 128 144
DarkSlateGray	2F 4F 4F	47 79 79
Black	00 00 00	0 0 0

6.1.1 Preliminaries

Before starting to proceed with the problem itself, we ought to mention about some problem specific concepts.

Real World Similarity

As stated in section 6.1 in the *RGB* scheme, all colors are represented by triplets of *Red*, *Green* and *Blue* values. With this in mind, when observing the similarity value between two colors, directly from the real-word data, we may consider them as if they were placed in *3D space*. Thus each of the *RGB* values acts as a coordinate for the corresponding color, and we may utilize a geometric distance formulation for this problem. In this scenario we adopt the *Euclidean distance*, which is indeed one of the common definitions in the field of *color science*. [Sha02]

In Euclidean three-space, the distance between points (x_1, y_1, z_1) and (x_2, y_2, z_2) is

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Thus in our problem, the similarity distance between two colors C and S follows as:

$$similarity\_distance = \sqrt{(C_{Red} - S_{Red})^2 + (C_{Green} - S_{Green})^2 + (C_{Blue} - S_{Blue})^2}$$

We then follow by normalizing this metric distance value with the following algorithm

$$degree\_of\_similarity = 1 - \frac{similarity\_distance}{\sqrt{3 * 255^2}}$$

so that the resulting values fall between the *real number* interval of $[0, 1]$.

As one will notice the denominator of the function is the highest possible distance in the metric space. Real world correspondence of this scenario consists the color pair of *Black* and *White*. The algorithm would return 0 as a result for this case. Moreover as expected, the pairing of any color with itself would result the algorithm computing 1 as the similarity degree.

Evaluated Similarity

The gist of our methodology, which is introduced in chapter 4, is conserved in the practical case with minor modifications.

The predicate trees are relatively simple in structural terms. All are depth one, have three children.

As we know main focus of our algorithm is utilizing predefined similarity relations between the subconcepts. In this particular example, we assume the only predefined similarity relations are the ones between the same RGB main colors. So for every Red_α and Red_β a similarity relations is defined where α and β are two integer values from the interval $[0, 255]$.

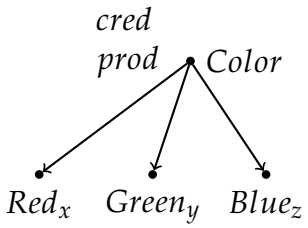


Figure 6.1: Example of a color predicate tree

The corresponding function is as follows:

$$similarity\_single\_color = 1 - \frac{|c_1 - c_2|}{255}$$

which informally equals to the normalization of the metric distance on a single coordinate of the colors, so that it falls into the interval of $[0, 1]$.

Lastly, the evaluation function follows the same procedure described in chapter 4.

Pseudo Credibility Values

As discussed in chapter 5, the main focus of interest is finding the weight constants in the *Hybrid Approach(HA)* in an automatic sense. As one might observe, the problem consists the necessary information for calculating the values of *VA* and *EA* thus for fixing the weights, solely the credibility value should also be known. So for training weights, there is a need for some initial *pseudo credibility values*.

In the end we want the product of our similarity and credibility estimation, close to the real degree of similarity.

$$degree\_of\_similarity \times degree\_of\_credibility \cong real\_similarity$$

Through the preceding sections, we know evaluating both of values concerning similarities. So by putting these values in the equation, we may compute the variable of interest.

Computing Weights

In section 5.0.8 we introduce the formula for the *Hybrid Approach* as follows:

$$Credibility = (w \times \textbf{[Vertex\_Approach]}) + ((1 - w) \times \textbf{[Edge\_Approach]}) \quad (6.1)$$

where $w \in [0..1]$.

As mentioned earlier, when one has the value of credibility, getting the value of the weights from the formula is trivial.

One noteworthy concept here is the *Global Weight*. The focus of this methodology is training the weights from some pre-given info, then fixing the weights through some procedure so the credibility values for new fuzzy rules can be determined.

For maintaining the Multi-Adjoint properties as shown in [MHPCS10], we adopt the minimum operator for evaluating the global weight. Informally, we train a single weight for every similarity pair in our training set, then set the minimum of those values as our global weight.

The Credibility

When the weights are computed, the credibility values are easily evaluated via the function we introduce in section 5.0.8. This consists the last step of the methodology, which could just be followed by checking both the credibility value and the similarity value together in comparison to the real degree of similarity.

Here the desired outcome is getting a conservative results, so to speak a value which does not exceed the original degree of similarity, but still finding one which is relatively close.

6.2 A Practical Example

Suppose we have the following six color pairings in our training set:

Listing 6.1: The training set

```
Salmon Coral                                1
Tomato Peru                                 2
YellowGreen MediumAquamarine                3
FineBrick LightSeaGreen                     4
Crimson RoyalBlue                           5
Snow MidnightBlue                           6
```

For the sake of an interesting display, let us select a pairing from the training set and one completely outside of the initial knowledge base, in order to construct our test set:

Listing 6.2: The training set

```
Tomato Peru                                 1
Pink Violet                                 2
```

With these information in hand, we are ready for pursuing with the calculations.

6.2.1 Training Process for Salmon and Coral

In this section we show step by step the process of training a weight via utilizing the info of a color pairing from the training set. The task is simply following the explicit process described in section 6.1.1.

The first step is finding the real-world similarity proximity between the colors, namely *Salmon* and *Coral*. And for that, one needs to find out the metric distance of similarity of the colors.

We input the corresponding *RGB* values of the colors into the modified euclidian function, defined in section 6.1.1.

$$similarity\_distance = \sqrt{(C_{Red} - S_{Red})^2 + (C_{Green} - S_{Green})^2 + (C_{Blue} - S_{Blue})^2}$$

$$\begin{aligned} similarity\_distance &= \sqrt{(C_{Red} - S_{Red})^2 + (C_{Green} - S_{Green})^2 + (C_{Blue} - S_{Blue})^2} \\ &= \sqrt{(C_{250} - S_{255})^2 + (C_{128} - S_{127})^2 + (C_{114} - S_{80})^2} \\ &\cong \mathbf{34.38} \end{aligned} \tag{6.2}$$

Then use this metric distance value in the similarity evaluation function:

$$\begin{aligned} degree\_of\_similarity &= 1 - \frac{similarity\_distance}{\sqrt{3 * 255^2}} \\ &= 1 - \frac{34.48}{\sqrt{3 * 255^2}} \\ &\cong \mathbf{92.21} \end{aligned} \tag{6.3}$$

Second step of the process is finding the degree of similarity between the colors via using our own evaluation algorithm. As we know first thing to do here is handling with the calculation of the similarity proximities of subconcepts. There will be three such relations that have to be taken into account. Let us compute one of them, namely the one concerning the *Red* value of the colors:

$$\begin{aligned} similarity\_single\_color &= 1 - \frac{|c_1 - c_2|}{255} \\ &= 1 - \frac{|250 - 255|}{255} \\ &\cong \mathbf{0.98} \end{aligned} \tag{6.4}$$

After doing the same computation for the subconcept similarity relations regarding *Green* and *Blue*, and putting them in the evaluation function, we get:

$$degree\_of\_similarity = 0.947$$

In section 6.1.1 we have stated that one we have the both results of the similarity evaluations, we may conclude the credibility value which will be used for training the weights. The corresponding formula and the computation is as follows:

$$\begin{aligned} degree\_of\_similarity \times degree\_of\_credibility &= real\_similarity \\ degree\_of\_credibility &= \frac{real\_similarity}{degree\_of\_similarity} \\ &= 1 - \frac{0.947}{0.98} \\ &\cong \mathbf{0.973} \end{aligned} \tag{6.5}$$

The last computation enables us to continue with the last step of training, which is evaluating the weight. In addition to the credibility value, the *VA* and *EA* values should also be calculated. Since all of the subconcepts are included in some similarity relation, the *VA* value is 1. Moreover as there are three predefined subconcept similarity relations out of nine possible relations(i.e. $3 \times 3 = 9$), the corresponding *EA* value is $1/3$.

With the help of all these information, the so-called *trained weight value* is:

$$\begin{aligned} \textit{Credibility} &= (w \times \textbf{[Vertex\_Approach]}) + ((1-w) \times \textbf{[Edge\_Approach]}) \\ w &= \frac{\textit{Credibility} - \textbf{[Edge\_Approach]}}{\textbf{[Vertex\_Approach]} - \textbf{[Edge\_Approach]}} \\ &= \frac{0.973 - 0.33}{1 - 0.33} \\ &\cong \mathbf{0.959} \end{aligned} \tag{6.6}$$

And this concludes our calculation for the first color pair of the training set.

6.2.2 The Outcome

A complete implementation of the process in *C++* programming language is displayed at the appendix.

Regarding this example, the console output of the program is depicted in the following page.

As stated in section 6.1.1, in order to preserve *Multi-Adjoint* properties, we take the *minumum* operation as the merging tool of the weight values from the training set.

By multiplying the credibility and the similarity values evaluated from our algorithm, we compare the resulting number with the real similarity degree.

As can be seen in both examples, particularly both in the case of the example from the training set and also the newly introduced color pair, the results prove to be conservative. In other words they do not exceed the real values, which is a desired feature. In addition to that, we observe that the evaluated values are relatively close to the real-word correspondences.

As both of these features are satisfied by the result of our methodology where a relatively small knowledge base is utilized, we consider our method to be displaying extreme promise for real world applications.

```
Please select your choice of program: a

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~TRAINING SET~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Color_pair[0], Salmon and Coral:
_____________________________________________
sim1:   0.947712        sim2:   0.922159
cred2:  0.973037        weight: 0.959555


Color_pair[1], Tomato and Peru:
_____________________________________________
sim1:   0.879739        sim2:   0.8615
cred2:  0.979268        weight: 0.968902


Color_pair[2], YellowGreen and MediumAquamarine:
_____________________________________________
sim1:   0.775163        sim2:   0.703893
cred2:  0.908058        weight: 0.862087


Color_pair[3], FineBrick and LightSeaGreen:
_____________________________________________
sim1:   0.443137        sim2:   0.44288
cred2:  0.999418        weight: 0.999128


Color_pair[4], Crimson and RoyalBlue:
_____________________________________________
sim1:   0.470588        sim2:   0.4525
cred2:  0.961562        weight: 0.942343


Color_pair[5], Snow and MidnightBlue:
_____________________________________________
sim1:   0.224837        sim2:   0.207335
cred2:  0.92216         weight: 0.88324


**************************************************
        Global weight: 0.862087
**************************************************


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~TEST SET~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Color_pair[6], Tomato and Peru:
_____________________________________________
sim1:   0.879739        sim2:   0.8615

cred1:  0.908058
simFin: 0.798854

**************************************************
        0.798854     =<     0.8615
**************************************************


Color_pair[7], Pink and Violet:
_____________________________________________
sim1:   0.85098         sim2:   0.83427

cred1:  0.908058
simFin: 0.77274

**************************************************
        0.77274      =<     0.83427
**************************************************
```

CHAPTER 7

CONCLUSIONS

The work presented here solves the problem of finding the similarity degree between two fuzzy predicates in the fuzzy logic domain. As mentioned in chapter 1, the motivation is fixing the imprecise nature of the technology that current web engines utilize. Once again informally, we introduce a framework where, when one queries a fuzzy concept such as *fast red car* the framework is able to make us of a crisp info such as the maximum speed of a particular car, and in the end returns a set of fuzzy answers. But on top of that, now the framework is able to retrieve answers regarding to similar queries like the ones for *fast orange car* or *average-speed red car*, with as expected lower credibility values. In this regard, a sound mathematical formalism has been introduced (see Chap. 4) which enabled the representation of predicates as graph trees. By that we are able to utilize an efficient search methodology as in the cases of graph problems, for detecting similar pairings of subconcepts. The last step is constructing an operational evaluation function that computes the resulting degree of similarity. (See Sec. 4.2.2)

The most recent approach in the field of interest is the *Structure Based Method(SBM)* [Lu11]. *SBM* makes use of the structural property by constructing the predicate tree via taking the head of a fuzzy rule as the root of the tree and by branching the tree regarding the body of the rule. Here one point of concern is that, the construction of the tree is recursive but not the evaluation of the similarity degree between the predicates. As a result of this the trees need to be structurally equivalent for the algorithm to be able to execute. This hole in the algorithm is patched by introducing identity predicates when one tree is lacking internal nodes or leaf nodes compared to the other one. But as Lu also confirms [Lu11], this causes a ”distortion” effect on the final evaluation. Especially in cases such as where the branching factors differ by a big margin or when one tree is highly unbalanced, this effect would be clearly apparent.

The main idea behind this work is introducing a methodology that avoids the shortcomings of the preceding approach. These points of problems and the way we tackle with them are inspected and demonstrated in detail in their corresponding sections.

- In order to evaluate the degree of similarity between two predicates, *SBM* requires their corresponding predicate tress to have the same graphical structure. Since in most scenarios, this indeed is not going to be the case, the missing branches of the trees are expanded with *identity predicates*. Moreover for the similarity evaluation function to operate, a default similarity proximity value is defined between an arbitrary predicate and the identity predicate. Assume the cases where there is a relatively big branching factor difference between the two predicate trees of interest. In such an example as discussed earlier, the smaller tree is going to be filled in the appropriate places with identity predicates. Thus most of the similarity values of predicate pairings are going to come from the default similarity rule which is defined for the identity predicate and so the final computation will resemble that value inevitably. As mentioned in section 3.2.2, in *SBM* as *the missing number of nodes in one tree increases, the similarity degree calculated by the algorithm converges to the default values* that is defined between an arbitrary predicate and the identity predicate.

◇ The justification of our algorithm avoiding this mishap is pretty straightforward. Since it does not introduce any such external knowledge, and just use the original information of the initial knowledge base, no such shortcomings are encountered in any scenarios. (See Sec. 4.3.2)

- After every level of expansion of the predicate trees, *SBM* checks every predicate pair between the two trees with respect to their resulting similarity degree. Before the next level of expansion is pursued, a filtering is done on the tree by selecting the best pair combination on the level. The problem with this approach is that the information on a prior level is incomplete, and thus wrong steps can be taken when filtering that will cause the loss of crucial information for the main focus, *i.e.* comparing the similarity values of the main predicates. (See Sec. 3.2.3)

◇ Once again, as the predicate trees are maintained in their original forms, the algorithm is able to work effectively in a manner that resembles *Depth First Search*, on the complete structure emitting the need of filtering some parts of the data. (See Sec. 4.3.3)

- The way that *SBM* utilizes the credibility values of the fuzzy rules in the similarity evaluation function for the predicates, proves to be problematic when the branching factor of the predicate tree is small. As the unit distance of the credibility values is directly summed with the values of similarity pairs in the numerator of the equation, when the cardinality n is small, the values of credibilities simply overshadow the values of the similarity pairs. (See Sec. 3.2.1) For instance, in the case where there is a single subconcept pairing, the effect of the credibility values over the final result would be exactly fifty percent which is highly excessive.

◇ In our approach unit distance of the credibility values of the fuzzy rules is not thrown in between the similarity proximity values of the pairings. They are

maintained separately, until the similarity values are processed thoroughly in the evaluation function. And only in the end utilized via the operator that is defined by the fuzzy rules themselves. (See Sec. 4.3.1)

- Yet another shortcoming of *SBM* is that the method is not able to utilize the information which comes from shared similarity concepts in similarity predicate pairings. The method tries to fix the pairings in an analogous manner of a cartesian product. In other words every subconcept is only mapped to another subconcept from the other predicate tree, thus only one of the fuzzy similarity relations where a particular subconcept appears is taken into account. (See Sec. 3.2.4)

  ◇ In very brief terms, our methodology is not bounded with such limitations as the preceding one. A particular subconcept may appear in as many fuzzy similarity rules as it may, and all of these information is realized and processed as the method is able to search and collect all of the similarity relations distinctly without one affecting the other. (See Sec. 4.3.4)

- *SBM* has no means for representing similarity relations as fuzzy rules. In other terms, all of the similarity relations that the approach contains are solely facts.

  ◇ In 4.3.5, we introduce sound syntax and semantics for representing similarity relations with credibility values. This extension enables us to investigate another topic of research, namely evaluating confidence values for computations of similarity degrees between two fuzzy predicates. (see Chap. 5)

It is only fair to also mention the shortcomings of our approach. Remember that when evaluating the final degree of similarity between two predicates, the predefined similarity relations on subconcepts are taken into account in the evaluation algorithm. Whereas in real world scenarios, one may encounter knowledge bases which do not contain complete information. In order to take into account such cases, we introduce three approximation methods concerning the confidence that we attain to the result of our similarity evaluation methodology over fuzzy predicates.

- The first approach is a naive one called the *Vertex Approach*, where the percentage of the subconcepts which appear in a similarity relation is computed with respect to the total number of subconcepts in the knowledge base. (See Sec. 5.0.6) The calculations of *Vertex Approach* prove to be highly plausible. However the problem of the method is that since it's only concerned with the cardinality of the subconcepts participating in similarity relations, or vertices in graph terms, in the scenario where more relations between the same set of subconcepts are introduced, it is not able to utilize the newly added information.

- The intuition behind the second approach, namely *Edge Approach*, is eliminating the shortcoming of the preceding one. This time the percentage of the similarity pairings between the subconcepts is computed with respect to the total number of potential pairings in the knowledge base. Speaking in graph terms, the latter

equates counting the number of added edges between the two predicate trees. Yet in this approach since the main parameter(i.e. similarity pairings) is normalized by a relatively large-scale parameter (i.e. potential pairings), the evaluated credibility values are dramatically lower than the expected ones.

- The source of motivation of the last approach is gathering beneficial characteristics of the first two approaches, whereas avoiding the shortcomings. In the light of this idea we propose the *Hybrid Approach* which merges two approaches together by attaining them some weights. For a given knowledge base and a similarity pairing as a result of the similarity evaluation algorithm, the methodology is able to compute a credibility value which is plausible and at the same time can realize all of the information presented by the knowledge base.

On top of these we introduce firstly a naive intuitionistic approach which is then followed by an atomized process of determining the weights in the credibility evaluation equation, in a similar manner how the problem is realized in [MHPCS10]. In chapter 6 we realize this method by training the system with the original knowledge base, and then using the evaluated weights in order to compute the credibility values of the similarity relations. We have been highly successful with the application in the sense that we have computed values which are extremely close to their corresponding real-world numbers. Moreover, we have evaluated conservative results, in other terms values which do not exceed the original degrees of similarity. As discussed in chapter 6, this notion holds great value for maintaining *Multi-Adjoint* properties. These two features depict that our methodology holds great promise for the real-life applications.

The current direction of the research is looking into other scientific fields from where we may find a new inspiration for a better methodology. One potential candidate that stands out is the set of sound and efficient algorithms used in artificial neural networks.

BIBLIOGRAPHY

[AG58] A.J.Cain and G.A.Harrison. An analysis of taxonomist's judgement of affinity. *Proc.Zool.Soc.Lond*, 131:85–98, 1958.

[Aga] Pragya Agarwal. Lotfi zadeh: Fuzzy logic-incoporating real-world vagueness. *Spatially Integrated Social Science's collection*, page 68.

[A.J69] A.J.Boyce. Mapping diversity: A comparative study of numerical methods. *Numerical taxonomy*, pages 1–31, 1969.

[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.

[A.T77] A.Tversky. Features of similarity. *Psychol.Rev*, 84:327–352, 1977.

[BC81] J. Barwise and R. Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219, 1981.

[BCW] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. Feedbackbypass: A new approach to interactive similarity query processing. *DEIS – CSITE-CNR and Microsoft Corp*.

[BH] Alexander Budanitsky and Graeme Hirst. Semantic distance in wordnet: An experimental application-oriented evaluation of five measures. *Department of Computer Science University of Toronto*.

[BMP95] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. Fril- fuzzy and evidential reasoning in artificial intelligence. 1995.

[CRARD08] Rafael Caballero, Mario Rodríguez-Artalejo, and Carlos A. Romero-Díaz. Similarity-based reasoning in qualified logic programming. In *Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming*, PPDP '08, pages 185–194, New York, NY, USA, 2008. ACM.

[DP03] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge Univ Press, 2003.

[ETC95] Susana Cubillo Enric Trillas and Juan Luis Castro. Conjunction and disjunction on ([0,1],≤). *Fuzzy set and systems*, 72(2):155–165, 1995.

[F.A50] F.Attneave. Dimensions of similarity. *American Journal of Psychology*, 63:516–556, 1950.

[FBF89] K. Forbus and D. Gentner. B. Falkenhainer. The structure mapping engine: algorithms and examples. *Artificial Intelligence*, 41(1):1–63, 1989.

[Fel98] Christiane Fellbaum. *WordNet An electronic lexical database*. MIT Press, 1998.

[GA59] G.Young and A.S.Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3:19–22, 1959.

[GAI+97] Ingo Glockner, Technischen Fakult At, Abteilung Informationstechnik, Impressum Herausgeber, Robert Giegerich, Alois Knoll, Helge Ritter, Gerhard Sagerer, Ipke Wachsmuth, Ingo Glockner, and Ingo Glockner. Dfs - an axiomatic approach to fuzzy quantification, 1997.

[GAOC] Adolfo Guzman-Arenas and Jesus M. Olivares-Ceja. Finding the most similar concepts in two different ontologies. *Centro de Investigación en Computación Instituto Politécnico Nacional*.

[GMHV04] S. Guadarrama, S. Munoz-Hernandez, and C. Vaucheret. Fuzzy prolog: a new approach using soft constraints propagation. *Fuzzy Sets and Systems (FSS)*, 144(1), 2004.

[GMV04] S. Guadarramaa, S. Munoz, and C. Vaucheret. Fuzzy prolog: a new approach using soft constraints propagation. *Fuzzy Sets and Systems*, 144:127–150, 2004.

[H.G46] H.Gulliksen. Paired comparison and the logic of measurement. *Psychological.Review*, 55:199–213, 1946.

[IK85] Mitsuru Ishizuka and Naoki Kanai. *Prolog-elf incorporating fuzzy logic*. Morgan Kaufmann Publishers Inc, 1985.

[Jac08] P. Jaccard. Nouvells researches sur la distribution florale. *Bulletin de la Societe de Vaud des Sciences Naturelles*, 44:223, 1908.

[JIRM11] Pascual Julián-Iranzo and Clemente Rubio-Manzano. A sound semantics for a similarity-based logic programming language. In *Proceedings of the 11th international conference on Artificial neural networks conference on Advances in computational intelligence - Volume Part II*, IWANN'11, pages 421–428, Berlin, Heidelberg, 2011. Springer-Verlag.

[JIRMGC09] Pascual Julián-Iranzo, Clemente Rubio-Manzano, and Juan Gallardo-Casero. Bousi prolog: a prolog extension language for flexible query answering. *Electron. Notes Theor. Comput. Sci.*, 248:131–147, August 2009.

[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.

[KS86] E. Keenan and J. Stavi. A semantic characterization of natural language determiners. *Linguistics and Philosophy*, 9, 1986.

[Kun80] Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs*. North-Holland, 1980.

[Lee72] R. C. T. Lee. Fuzzy logic and the resolution principle. *the Association for Computing Machinery*, 19(1):119–129, 1972.

[LL90] Deyi Li and Dongbo Liu. A fuzzy prolog database system. 1990.

[Llo87] John Wylie Lloyd, editor. Springer, 1987.

[Lu.79] S. Y. Lu. A tree-to-tree distance and its application to cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 1979.

[Lu11] Jingwei Lu. Extending fuzzy logic with characteristics similarity and quantification. Master's thesis, Facultad de Informatica, 2011.

[Luk20] Jan Lukasiewicz. *On three-valued logic*. North Holland, 1920.

[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.

[MT92] Jiri Matousek and R. Thomas. *On the complexity of finding iso- and others morphisms for partial k-trees*. 1992.

[M.W38] Richardson M.W. Multidimensional psychophysics. *Psychol.Bull*, 38:659, 1938.

[NW00] H.T. Nguyen and E.A. Walker. *A first Course in Fuzzy Logic*. Chapman and Hall/CRC, 2000.

[P.98] Hajek P. *Metamathematics of Fuzzy Logic*. Dordrecht: Kluwer, 1998.

[RP63] R.R.Sokal and P.H.Sneath. *Principles of Numerical Taxonomy*. Freeman San Francisco CA, 1963.

[SA95] Victor Vianu Serge Abiteboul, Richard Hull. *Foundations of databases*. Addison Wesley Longman, 1995.

[Ses02] Maria I. Sessa. Approximate reasoning by similarity-based sld resolution. *Theor. Comput. Sci.*, 275(1-2):389–426, March 2002.

[Sha02] Gaurav Sharma. *Digital Color Imaging Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 2002.

[T.02] Andreasen T. On knowledge-guided fuzzy aggregation. *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2002.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math*, 5:285–309, 1955.

[VA02] Valerie V.Cross and Thomas A.Sudkamp. *Similarity and Compatiability in Fuzzy Set Theory*. Springer-Verlag company, 2002.

[vB83] J. van Benthem. Determiners and logic. *Linguistics and Philosophy*, 6, 1983.

[vB84] J. van Benthem. Questions about quantifiers. *Journal of Symbolic Logic*, 49, 1984.

[Wri08] Craig S. Wright. *The IT Regulatory and Standards Compliance Handbook:: How to Survive Information Systems Audit and Assessments*. Syngress Publishing, 2008.

[W.S65] W.S.Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1965.

[Zad83] L.A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computers and Mathematics with Applications*, 9:149–184, 1983.

[Zad88] Lofti A. Zadeh. *Fuzzy Logic*. Stanford University. Center for the Study of Language and Information, 1988.

[Zad96] L.A. Zadeh. The birth and evolution of fuzzy logic. *Advances in fuzzy systems*, 6:811–814, 1996.

[ZM89] L.Ding Z.Shen and M.Mukaidono. Fuzzy resolution principle. *18th International Symposium on Multiple-valued Logic*, 5, 1989.

APPENDIX

Here we display the implementation of the methodology that is discussed in detail in chapter 6.

Firstly we should briefly mention about the flow of the program and the structure of the input files.

The main program contains of two subprograms. The first one of those takes two input files, namely *colorNames.txt* and *colors.txt*. The program takes a used-defined number of color pairs. The last two pairs are the test set, and the rest of the pairs consist the training set. One of the two pairings of the test set is expected to be from the training set, and the other one outside of the domain of the training set. So the program trains the weights via the color pairings from the training set and evaluate the credibility value of the one from the test set. For this regard, *colorNames.txt* store the information concerning the names of the colors. Every line displays one pair, and in each pair colors are separated by the *space* escape character. In a similar manner, in each line *colors.txt* saves the *RGB* values of the corresponding colors from the other input file. The six values, i.e. three for each of the colors, are separated by *space* characters and lines are split with an *enter key*.

An example case for the input files is displayed as follows:

Listing 7.1: colors.txt

```
250 128 114 255 127 80                                    1
255 99 71 210 105 30                                      2
154 205 50 102 205 170                                    3
178 34 34 32 178 170                                      4
220  20  60 65 105 225                                    5
255 250 250 25 25 112                                     6
255 99 71 210 105 30                                      7
255 192 203 238 130 238                                   8
```

Listing 7.2: colorNames.txt

```
Salmon Coral                                    1
Tomato Peru                                     2
YellowGreen MediumAquamarine                    3
FineBrick LightSeaGreen                         4
Crimson RoyalBlue                               5
Snow MidnightBlue                               6
Tomato Peru                                     7
Pink Violet                                     8
```

The second subprogram works with a single input file, namely *colorsAll.txt*. The input file contains all the information regarding the complete set of colors in *X11* scheme. At each line you have the name and the *RGB* values of a particular color.

The subprogram asks the user two name two colors from the domain. It then takes out those two values and their associated information from the knowledge base. All of the possible pairings between the rest of the colors are computed by the program, which exactly corresponds to 9591 pairings. With this huge amount of information, the weights are trained and later utilized for evaluating the credibility degree of the similarity value concerning the color pairing of interest.

You may observe an example instance of the *colorsAll.txt* input file. That is followed by the complete implementation of the methodology in *C++* programming language.

Listing 7.3: colorsAll.txt

```
IndianRed 205 92 92                                1
LightCoral 240 128 128                             2
Salmon 250 128 114                                 3
DarkSalmon 233 150 122                             4
LightSalmon 255 160 122                            5
Red 255 0 0                                        6
Crimson 220 20 60                                  7
FireBrick 178 34 34                                8
DarkRed 139 0 0                                    9
Pink 255 192 203                                   10
LightPink 255 182 193                              11
HotPink 255 105 180                                12
DeepPink 255  20 147                               13
MediumVioletRed 199  21 133                        14
PaleVioletRed 219 112 147                          15
LightSalmon 255 160 122                            16
Coral 255 127  80                                  17
Tomato 255  99  71                                 18
OrangeRed 255  69   0                              19
DarkOrange 255 140   0                              20
Orange 255 165   0                                 21
Gold 255 215   0                                   22
Yellow 255 255   0                                 23
LightYellow 255 255 224                            24
LemonChiffon 255 250 205                           25
LightGoldenrodYellow 250 250 210                   26
PapayaWhip 255 239 213                             27
Moccasin 255 228 181                               28
PeachPuff 255 218 185                              29
PaleGoldenrod 238 232 170                          30
Khaki 240 230 140                                  31
DarkKhaki 189 183 107                              32
Lavender 230 230 250                               33
Thistle 216 191 216                                34
Plum 221 160 221                                   35
Violet 238 130 238                                 36
Orchid 218 112 214                                 37
Fuchsia 255   0 255                                38
Magenta 255   0 255                                39
MediumOrchid 186  85 211                           40
MediumPurple 147 112 219                           41
BlueViolet 138  43 226                             42
DarkViolet 148   0 211                             43
DarkOrchid 153  50 204                             44
DarkMagenta 139   0 139                           45
Purple 128   0 128                                 46
Indigo  75   0 130                                 47
DarkSlateBlue 72  61 139                           48
SlateBlue 106  90 205                              49
MediumSlateBlue 123 104 238                        50
GreenYellow 173 255  47                            51
Chartreuse 127 255   0                             52
LawnGreen 124 252   0                              53
```

```
Lime 0 255   0                                  54
LimeGreen  50 205  50                           55
PaleGreen 152 251 152                           56
LightGreen 144 238 144                          57
MediumSpringGreen 0 250 154                     58
SpringGreen 0 255 127                           59
MediumSeaGreen 60 179 113                       60
SeaGreen 46 139  87                             61
ForestGreen 34 139  34                          62
Green 0 128   0                                 63
DarkGreen 0 100   0                             64
YellowGreen 154 205  50                         65
OliveDrab 107 142  35                           66
Olive 128 128   0                               67
DarkOliveGreen 85 107  47                       68
MediumAquamarine 102 205 170                    69
DarkSeaGreen 143 188 143                        70
LightSeaGreen 32 178 170                        71
DarkCyan  0 139 139                             72
Teal  0 128 128                                 73
Aqua 0 255 255                                  74
Cyan 0 255 255                                  75
LightCyan 224 255 255                           76
PaleTurquoise 175 238 238                       77
Aquamarine 127 255 212                          78
Turquoise 64 224 208                            79
MediumTurquoise 72 209 204                      80
DarkTurquoise 0 206 209                         81
CadetBlue 95 158 160                            82
SteelBlue 70 130 180                            83
LightSteelBlue 176 196 222                      84
PowderBlue 176 224 230                          85
LightBlue 173 216 230                           86
SkyBlue 135 206 235                             87
LightSkyBlue 135 206 250                        88
DeepSkyBlue 0 191 255                           89
DodgerBlue 30 144 255                           90
CornflowerBlue 100 149 237                      91
RoyalBlue 65 105 225                            92
Blue 0   0 255                                  93
MediumBlue 0   0 205                            94
DarkBlue 0   0 139                              95
Navy 0   0 128                                  96
MidnightBlue 25  25 112                         97
Cornsilk 255 248 220                            98
BlanchedAlmond 255 235 205                      99
Bisque 255 228 196                              100
NavajoWhite 255 222 173                         101
Wheat 245 222 179                               102
BurlyWood 222 184 135                           103
Tan 210 180 140                                 104
RosyBrown 188 143 143                           105
SandyBrown 244 164  96                          106
Goldenrod 218 165  32                           107
DarkGoldenrod 184 134  11                       108
```

```
Peru 205 133  63                                109
Chocolate 210 105  30                           110
SaddleBrown 139  69  19                         111
Sienna 160  82  45                              112
Brown 165  42  42                               113
Maroon 128   0   0                              114
White 255 255 255                               115
Snow 255 250 250                                116
Honeydew 240 255 240                            117
MintCream 245 255 250                           118
Azure 240 255 255                               119
AliceBlue 240 248 255                           120
GhostWhite 248 248 255                          121
WhiteSmoke 245 245 245                          122
Seashell 255 245 238                            123
Beige 245 245 220                               124
OldLace 253 245 230                             125
FloralWhite 255 250 240                         126
Ivory 255 255 240                               127
AntiqueWhite 250 235 215                        128
Linen 250 240 230                               129
LavenderBlush 255 240 245                       130
MistyRose 255 228 225                           131
Gainsboro 220 220 220                           132
LightGrey 211 211 211                           133
Silver 192 192 192                              134
DarkGray 169 169 169                            135
Gray 128 128 128                                136
DimGray 105 105 105                             137
LightSlateGray 119 136 153                      138
SlateGray 112 128 144                           139
DarkSlateGray  47  79  79                       140
Black 0   0   0                                 141
```

Listing 7.4: The C++ implementation of the application concerning colors

```
                                                                                1
#include<iostream>                                                              2
using namespace std;                                                            3
#include<string>                                                                4
#include<math.h>                                                                5
#include <vector>                                                               6
#include <string>                                                               7
#include <fstream>                                                              8
#include <sstream>                                                              9
                                                                                10
struct color{                                                                   11
        int x;                  //first coordinate                              12
        int y;                  //second coordinate                             13
        int z;                  //third coordinate                              14
        string c;               //the color name                                15
};                                                                              16
                                                                                17
float euqDist(float, float, float, float, float, float);        ↩               18
          //euclidian distance between two colors.
float simOneColor(float, float);                                ↩               19
                                        //similarity between ↩
   two instances of the same color.
float simByED(float);                                           ↩               20
                                                  //similarity ↩
   degree of two colors via euqlidian distance.
float simByAlg(float, float, float, float, float, float);       ↩               21
   //similarity degree of two colors via our algorithm.
float credBySim(float, float);                                  ↩               22
                                    //credibility value defined ↩
   from proximity of similarity values.
float weightEval(float, float= 1, float = (1/3.0));             ↩               23
                  //weight w, evaluated from cred function.
float credEval(float, float= 1, float = (1/3.0));               ↩               24
                  //final credibility value, computed via global ↩
   weight
void inputHandle(ifstream& f,  vector<vector<float>> &c, string ↩               25
   fileName);
void inputColor(ifstream& f,  vector<string> &col, string fileName);            26
void inputComplete(ifstream& f,  vector<color> &c, string fileName);            27
                                                                                28
int main()                                                                      29
{                                                                               30
        string pc;                                                              31
        begin:                                                                  32
        cout<<"Please␣select␣your␣choice␣of␣program:␣";                         33
        cin>>pc;                                                                34
                                                                                35
        if(pc == "1" || pc == "A" || pc == "a"){                                36
                ifstream f1, f2;                                                37
                vector<vector<float>> colorPairs;                               38
                inputHandle(f1, colorPairs, "colors.txt");                      39
                vector<string> colorNames;                                      40
                inputColor(f2, colorNames, "colorNames.txt");                   41
```

```
cout<<"\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
cout<<"~~~~~~~~~~~~~~~~~~~~TRAINING SET~~~~~~~~~~~~~~~~~~~~\n";
cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

float weightMin = 1;
for(int i = 0; i < colorPairs.size() - 2; i++){
        float sim1 = simByAlg(colorPairs[i][0], colorPairs[i][1], colorPairs[i][2], colorPairs[i][3], colorPairs[i][4], colorPairs[i][5]);
        float sim2 = simByED(euqDist(colorPairs[i][0], colorPairs[i][1], colorPairs[i][2], colorPairs[i][3], colorPairs[i][4], colorPairs[i][5]));
        float cred2 = credBySim(sim1, sim2);
        float weight = weightEval(cred2);
        if(weight < weightMin)
                weightMin = weight;

        cout<<"Color_pair["<<i<<"], "<<colorNames[i*2]<<" and "<<colorNames[i*2+1]<<": "<<endl;
        cout<<"_________________________________________"<<e
        cout<<"sim1:\t"<<sim1<<"\tsim2:\t"<<sim2<<"\ncred2:\
        \tweight:\t"<<weight<<"\n\n\n";
}

cout<<"************************************************\n";
cout<<"\tGlobal weight: "<<weightMin<<endl;
cout<<"************************************************\n\n";

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
cout<<"~~~~~~~~~~~~~~~~~~~~~~TEST SET~~~~~~~~~~~~~~~~~~~~~~\n";
cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

int s = colorPairs.size();
for(int t = s -2; t < s; t++){
    float sim1 = simByAlg(colorPairs[t][0], colorPairs[t][1], colorPairs[t][2], colorPairs[t][3], colorPairs[t][4], colorPairs[t][5]);
    float sim2 = simByED(euqDist(colorPairs[t][0], colorPairs[t][1], colorPairs[t][2], colorPairs[t][3], colorPairs[t][4], colorPairs[t][5]));
```

```
                cout<<"Color pair["<<t<<"], ↩                    73
                    "<<colorNames[t*2]<<" and ↩
                    "<<colorNames[t*2+1]<<": "<<endl;
                cout<<"----------------------------------------"<<endl;  74
                cout<<"sim1:\t"<<sim1<<" ↩                       75
                    \tsim2:\t"<<sim2<<"\n\n";
                                                                 76
                float cred1 = credEval(weightMin);      ↩        77
                    //global weight is used for finding the ↩
                    final credibility value
                cout<<"cred1:\t"<<cred1<<endl;                   78
                float simFin = cred1 * sim1;                     79
                cout<<"simFin:\t"<<simFin;                       80
                                                                 81
                cout<<"\n\n**********************************"<<82
                cout<<"\t"<<simFin<<"      =<      "<<↩          83
                    "<<sim2<<endl;
                cout<<"**********************************"<<84
        }                                                        85
}                                                                86
                                                                 87
else if(pc == "2" || pc == "B" || pc == "b"){                    88
        ifstream f3;                                             89
        vector<color> col;                                       90
        inputComplete(f3, col, "colorsAll.txt");                 91
        string userColor1, userColor2;                           92
                                                                 93
        cout<<"\nPlease choose two colors: ";                    94
        cin>>userColor1;                                         95
        cin>>userColor2;                                         96
                                                                 97
        color temp1, temp2;                                      98
        for(int i = 0; i < col.size(); i++){                     99
                if(col[i].c == userColor1){                      100
                        temp1.c = col[i].c;             ↩        101
                                        //store the ↩
                            color at a temp variable
                        temp1.x = col[i].x;                      102
                        temp1.y = col[i].y;                      103
                        temp1.z = col[i].z;                      104
                                                                 105
                        col[i].c = col[col.size()-1].c; ↩        106
                            //switch it with the last ↩
                            element of
                        col[i].x = col[col.size()-1].x; ↩        107
                            //the vector in order to pop it ↩
                            out
                        col[i].y = col[col.size()-1].y; ↩        108
                            //of the list
                        col[i].z = col[col.size()-1].z;          109
                        col.pop_back();                          110
                }                                                111
        }                                                        112
                                                                 113
        for(int i = 0; i < col.size(); i++){                     114
```

```
            if(col[i].c == userColor2){                                  115
                    temp2.c = col[i].c;                          ↩       116
                                        //store the ↩
                        color at a temp variable
                    temp2.x = col[i].x;                                  117
                    temp2.y = col[i].y;                                  118
                    temp2.z = col[i].z;                                  119
                                                                         120
                    col[i].c = col[col.size()-1].c; ↩                    121
                        //switch it with the last ↩
                        element of
                    col[i].x = col[col.size()-1].x; ↩                    122
                        //the vector in order to pop it ↩
                        out
                    col[i].y = col[col.size()-1].y; ↩                    123
                        //of the list
                    col[i].z = col[col.size()-1].z;                      124
                    col.pop_back();                                      125
            }                                                            126
    }                                                                    127
                                                                         128
    cout<<"\n Weights evaluated via Min or Average ↩                     129
        rule: ";
    cin>>pc;                                                             130
                                                                         131
    float weightGlo = 1.0;          //global weight                      132
    if(pc == "1" || pc == "m" || pc == "M"){                             133
            color t1, t2;                                                134
            int pn =1;                                   ↩               135
                        //pair counter
                                                                         136
            for(int i = 0; i < col.size(); i++){                         137
                    for(int j = i+1; j < col.size(); ↩                   138
                        j++){
                            float sim1 = ↩                               139
                                simByAlg(col[i].x, ↩
                                col[i].y, col[i].z, ↩
                                col[j].x, col[j].y, ↩
                                col[j].z);
                            float sim2 = ↩                               140
                                simByED(euqDist(col[i].x, ↩
                                col[i].y, col[i].z, ↩
                                col[j].x, col[j].y, ↩
                                col[j].z));
                            float cred2 = ↩                              141
                                credBySim(sim1, sim2);
                            float weight = ↩                             142
                                weightEval(cred2);
                            if(weight < weightGlo)                       143
                                    weightGlo = weight;                  144
                                                                         145
                            cout<<"Color pair["<<pn<<"], ↩               146
                                "<<col[i].c<<" and ↩
                                "<<col[j].c<<": "<<endl;
                            cout<<"-------------------------147----------
```

```
                        cout<<"sim1:\t"<<sim1<<"\tsim2:\t"<<sim2<<"\ncred2
                            \tweight:\t"<<weight<<"\n\n\n";
                        pn++;
                    }
                }
            }

            else if(pc == "a" || pc == "A" || pc == "2"){
                color t1, t2;
                vector<float> weights;
                int pn =1;
                    //pair counter
                for(int i = 0; i < col.size(); i++){
                    for(int j = i+1; j < col.size(); j++){
                        float sim1 = simByAlg(col[i].x, col[i].y, col[i].z, col[j].x, col[j].y, col[j].z);
                        float sim2 = simByED(euqDist(col[i].x, col[i].y, col[i].z, col[j].x, col[j].y, col[j].z));
                        float cred2 = credBySim(sim1, sim2);
                        float weight = weightEval(cred2);
                        weights.push_back(weight);

                        cout<<"Color pair["<<pn<<"], and "<<col[i].c<<" and "<<col[j].c<<": "<<endl;
                        cout<<" ------------------------------------------"<<
                        cout<<"sim1:\t"<<sim1<<"\tsim2:\t"<<sim2<<"\ncred2
                            \tweight:\t"<<weight<<"\n\n\n";
                        pn++;
                    }
                }
                for(int i = 0; i < col.size(); i++)
                    weightGlo += weights[i];
                weightGlo /= col.size();
            }

            cout<<"*********************************************\n";
            cout<<"\tGlobal weight: "<<weightGlo<<endl;
            cout<<"*********************************************\n\n";

            cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"~~~~~~~~~~~~~~~~~~~~TEST SET~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

            float sim1 = simByAlg(temp1.x, temp1.y, temp1.z,
```

```
                temp2.x, temp2.y, temp2.z);
            float sim2 = simByED(euqDist(temp1.x, temp1.y, ↩
                temp1.z, temp2.x, temp2.y, temp2.z));
            cout<<"Color_pair[focus],_"<<temp1.c<<"_and_↩
                "<<temp2.c<<":_"<<endl;
            cout<<"_____"<<endl;
            cout<<"sim1:\t"<<sim1<<"_\tsim2:\t"<<sim2<<"\n\n";

            float cred1 = credEval(weightGlo);      //global ↩
                weight is used for finding the final ↩
                credibility value
            cout<<"cred1:\t"<<cred1<<endl;
            float simFin = cred1 * sim1;
            cout<<"simFin:\t"<<simFin;

            cout<<"\n\n**********\n";
            cout<<"\t"<<simFin<<"_____=<_____"<<sim2<<endl;
            cout<<"**********\n\n";
        }

        else{
            cout<<"Wrong_command!\n";
            goto begin;
        }

        getchar();
        getchar();
        return 0;
}

//euclidian distance between two colors
float euqDist(float a, float b, float c, float x, float y, float z)
{
        return ↩
            sqrt(pow(fabs(a-x),2)+pow(fabs(b-y),2)+pow(fabs(c-z),2));
}

//similarity between two instances of the same color. (ex: red_90 ↩
    and red_187)
float simOneColor(float c1, float c2)
{
        return 1-((fabs(c1-c2))/255);
}

//similarity degree of two colors via euqlidian distance
float simByED(float dist)
{
        return 1-(dist/(255*sqrt(3.0)));
        /*
        float res = 1-(dist/(255*sqrt(3.0)));
        if(res < 0.1)
                return 0;
        else
                return res;*/
}
```

```
                                                                    234
//similarity degree of two colors via our algorithm                 235
float simByAlg(float a, float b, float c, float x, float y, float z) 236
{                                                                   237
        return (((simOneColor(a, x))+(simOneColor(b, ↩              238
           y))+(simOneColor(c, z)))/3);
}                                                                   239
                                                                    240
//credibility value defined from proximity of similarity values    241
float credBySim(float sim1, float sim2)                             242
{                                                                   243
        //return 1-(fabs(sim1 - sim2));                             244
        float sim = (sim2+0.01) / (sim1+0.01);  //in order to ↩     245
           prevent division by zero
        if(sim > 1)                                                 246
                return 1;                                           247
        else                                                        248
                return sim;                                         249
}                                                                   250
                                                                    251
//weight w, evaluated from cred function                            252
float weightEval(float cred, float va, float ea)              ↩     253
                    //default values for va and ea are 1 and 1/3 ↩
   respectively
{                                                             ↩     254
                                                                  ↩
          //since those are the most common cases, but could be ↩
   overwritten
        return ((cred-ea)/(va-ea));                                 255
}                                                                   256
                                                                    257
float credEval(float weight, float va, float ea)              ↩     258
                    //again default values for va and ea are 1 and ↩
   1/3 respectively
{                                                                   259
        return ((weight * va) + ((1-weight) * ea));                 260
}                                                                   261
                                                                    262
                                                                    263
void inputHandle(ifstream& f,  vector<vector<float>> &c, string ↩   264
   fileName)
{                                                                   265
        string temp;                                                266
        f.open(fileName);                                           267
                                                                    268
        if (!f) {                                                   269
        cout << "Unable to open file";                              270
        cin>>temp;                                                  271
        exit(1); // terminate with error                            272
        }                                                           273
                                                                    274
        int i = 0;                                                  275
        char line[1024];                                            276
                                                                    277
                                                                    278
```

```
        while(f.getline(line, 1024)){                                   279
                istringstream ss(line, istringstream::in);              280
                float rate;                                             281
                vector<float> temp;                                     282
                                                                        283
                c.push_back(temp);                                      284
                                                                        285
                for(int j = 0; j < 6; j++){                             286
                        ss >> rate;                                     287
                        c[i].push_back(rate);                           288
                }                                                       289
                i++;                                                    290
        }                                                               291
                                                                        292
        f.close();                                                      293
}                                                                       294
                                                                        295
void inputColor(ifstream& f,  vector<string> &c, string fileName)       296
{                                                                       297
        string temp;                                                    298
        f.open(fileName);                                               299
                                                                        300
        if (!f) {                                                       301
        cout << "Unable to open file";                                  302
        cin>>temp;                                                      303
        exit(1); // terminate with error                                304
        }                                                               305
                                                                        306
        char line[1024];                                                307
        string tempColor;                                               308
                                                                        309
        while(f.getline(line, 1024)){                                   310
                istringstream ss(line, istringstream::in);              311
                ss >> tempColor;                                  ↩     312
                   //first color of the pairing
                c.push_back(tempColor);                                 313
                ss >> tempColor;                                  ↩     314
                   //second color of the pairing
                c.push_back(tempColor);                                 315
        }                                                               316
                                                                        317
        f.close();                                                      318
}                                                                       319
                                                                        320
void inputComplete(ifstream& f,  vector<color> &c, string fileName)     321
{                                                                       322
        string temp;                                                    323
        f.open(fileName);                                               324
                                                                        325
        if (!f) {                                                       326
        cout << "Unable to open file";                                  327
        cin>>temp;                                                      328
        exit(1); // terminate with error                                329
        }                                                               330
                                                                        331
```

```
        char line[1024];                                                    332
                                                                            333
        while(f.getline(line, 1024)){                                       334
                istringstream ss(line, istringstream::in);                  335
                color temp;                                                 336
                                                                            337
                ss >> temp.c;                                               338
                ss >> temp.x;                                               339
                ss >> temp.y;                                               340
                ss >> temp.z;                                               341
                                                                            342
                c.push_back(temp);                                          343
                                                                            344
                cout<<temp.x<<" \t"<<temp.y<<" \t"<<temp.z<<" \↩            345
                    t"<<temp.c<<endl;
        }                                                                   346
                                                                            347
        f.close();                                                          348
}                                                                           349
```