Assignment 3 - Reasoning and Ontology Matching

Zefan Liang

School of Electrical Engineering and Computer Science

University of Ottawa

19th March, 2019

CSI 5180 Topics in AI - Ontologies and Semantic Web

Caroline Barrière

Table of content

1 Introduction

This report contains the whole content of Reasoning and Ontology Matching's requirements.

2 Description Logic and OWL

(a) .provide two examples of concepts or roles which are within the ALC

(1).2013 $\subseteq$ Bundesliga $\cup$ UEFA_Champions_League

Figure 1 - Show example in my ontology

(2).Stadium(Signal_Iduna_Park)

Figure 2 - Show example in my ontology

(b) provide two examples of concepts or roles which are NOT within the ALC (too expressive)

(1).Player hasHomeField value Signal_Iduna_Park

Figure 3 - Show example in my ontology

(2).2014 Win only Runner_up

Figure 4 - Show example in my ontology

(c) In Description Logics, we think of a Knowledge Base as being the combination of a TBox and a ABox. Can you give two examples from your ontology that would be part of the TBox, and two examples of instances you would put in the ABox.

T-BOX:{

Sports Manager $\subseteq$ People,

4-1-2-3A $\subseteq$ Team_Lineup

}

A-BOX:{

4-1-2-3A(Christian_Pulisic, Dan-Axel_Zagadou, Lukasz_Piszczek, Mahmoud_Dahoud,

Marcel_Schmelzer, Marco_Reus, Mario_Gotze, Maximilian_Philipp, Paco_Alcacer,

Roman_Burki),

Sports Manager(Michael_Zorc)

}

3 Reasoning with Tableaux Algorithm

Dietary KB

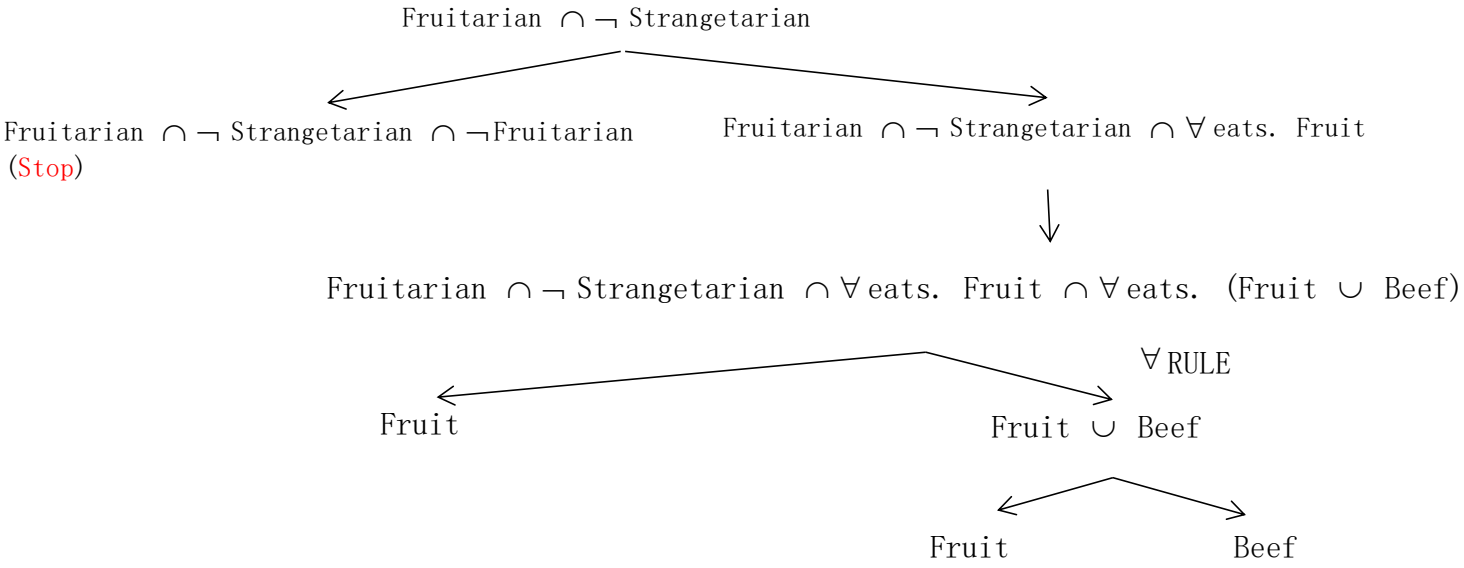
T-Box	MeatEater $\sqsubseteq \forall eats.$ (Chicken $\sqcup$ Beef) Vegetarian $\sqsubseteq \forall eats.\neg$Meat Vegan $\equiv$ Vegetarian $\sqcap \forall eats.$ ($\neg$RecipeWithDairy $\sqcap \neg$Dairy) Fruitarian $\sqsubseteq \forall eats.$ Fruit Strangetarian $\sqsubseteq \forall eats.$ (Fruit $\sqcup$ Beef) RecipeWithDairy $\equiv \exists madeWith.$Dairy IceCream $\sqsubseteq$ Dairy Chicken $\sqsubseteq$ Meat Beef $\sqsubseteq$ Meat
A-Box	madeWith(sundae, chocolate-ice-cream) IceCream(chocolate-ice-cream) Fruit(tangerine)

Figure 5 - the Dietary KB

Q1

Qyery:Fruitarian $\subseteq$ Strangetarian

Convert to Fruitarian $\cap \neg$ Strangetarian is unsatisfiable.



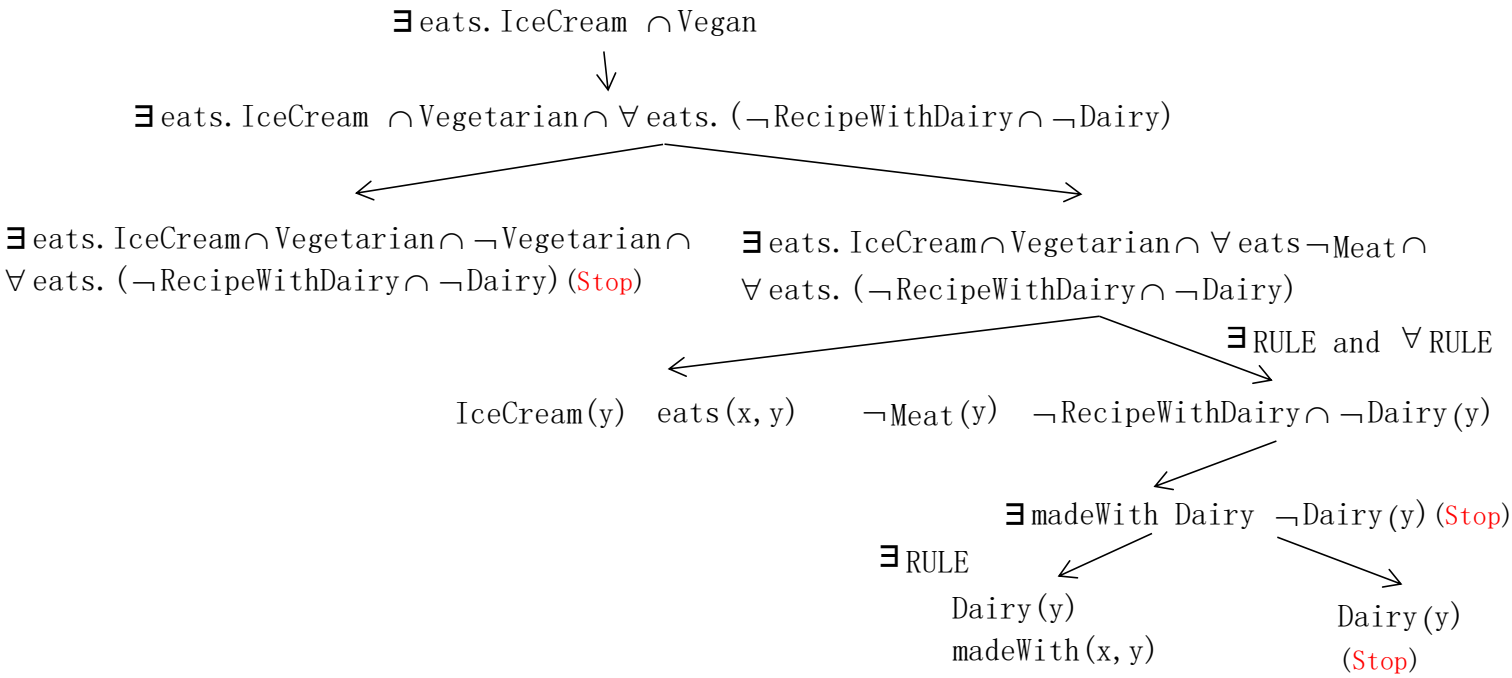
So Fruitarian $\cap \neg$ Strangetarian is satisfiable.

So Fruitarian $\subseteq$ Strangetarian is false.

Q2

Query: ∃ eats.IceCream ⊆ ¬ Vegan

Convert to ∃ eats.IceCream ∩ Vegan is unsatisfiable.



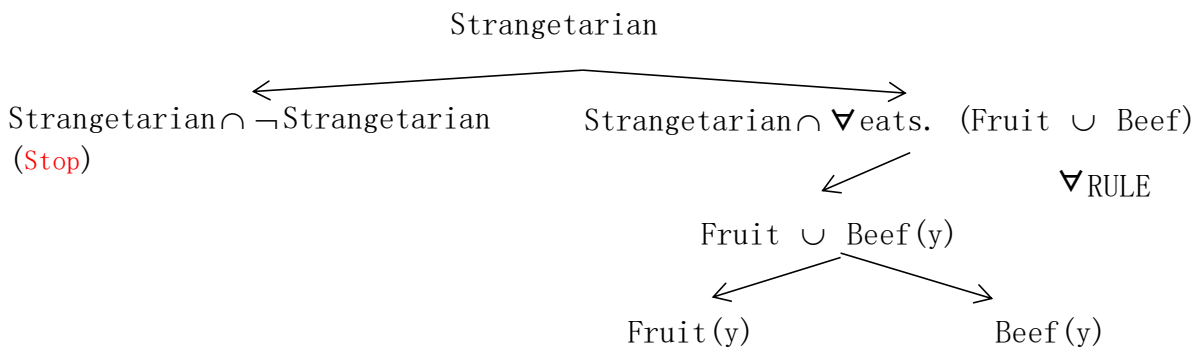
So ∃ eats.IceCream ∩ Vegan is satisfiable.

So ∃ eats.IceCream ⊆ ¬ Vegan is False.

Q3

Query:Strangetarian ≡ ⊥

Convert to Strangetarian is unsatisfiable.



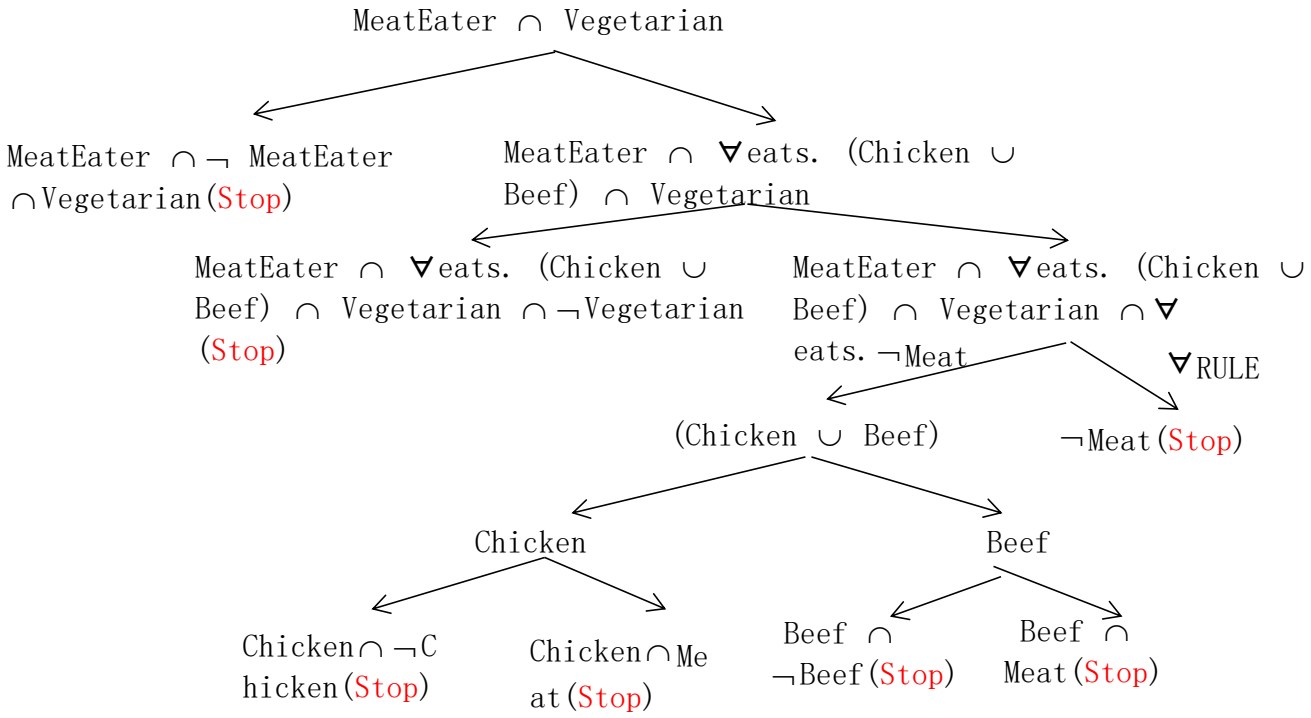
So Strangetarian ≡ ⊥ is satisfiable.

So Strangetarian ≡ ⊥ is False.

Q4

Query: MeatEater ∩ Vegetarian ≡ ⊥

Convert to MeatEater ∩ Vegetarian is unsatisfiable



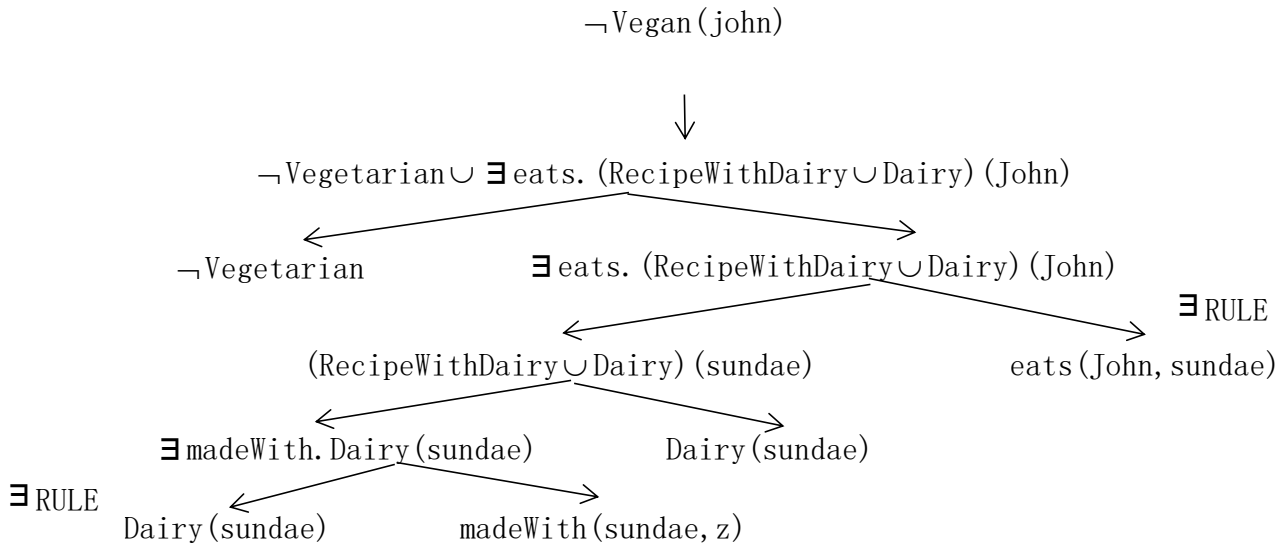
So Strangetarian ≡ ⊥ is unsatisfiable.

So Strangetarian ≡ ⊥ is True.

Q5 - eats(john, sundae), Vegan(john)

Query:Vegan(john)

Convert to ¬Vegan(john) is unsatisfiable.



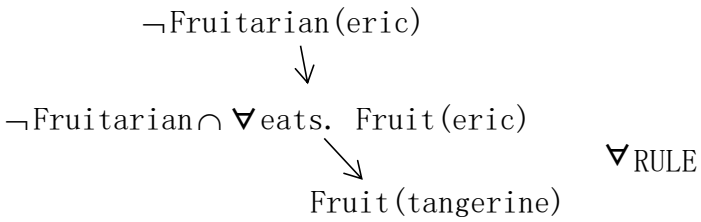
So ¬Vegan(john) is satisfiable.

So Vegan(john) is False.

Q6 eats(eric, tangerine), Fruitarian(eric)

Query:Fruitarian(eric)

Convert to ¬Fruitarian(eric) is unsatisfiable.



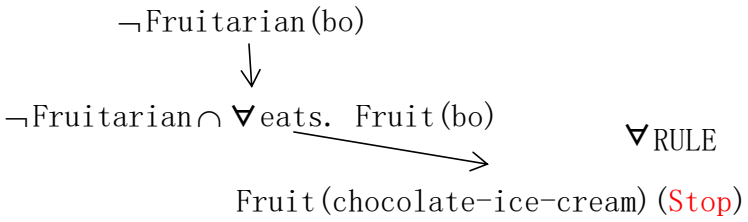
So ¬Fruitarian(eric) is satisfiable.

So Fruitarian(eric) is False.

Q7 eats(bo, chocolate-ice-cream), Fruitarian(bo)

Query:Fruitarian(bo)

Convert to ¬Fruitarian(bo) is unsatisfiable.



So ¬Fruitarian(bo) is unsatisfiable.

Fruitarian(bo) is True.

I think we should redefine Fruitarian as

Fruitarian $\subseteq \forall$ eats. Fruit $\cap \forall$ eats.(¬RecipeWithDairy $\cup$ ¬Dairy)

4 Investigating Reasoning tool

(a)Bossam is an inference engine for the semantic web. It is basically a RETE-based rule engine with native supports for reasoning over OWL ontologies, SWRL ontologies, and RuleML rules.

Here is the link:https://bossam.wordpress.com/about-bossam/

Bossom supports for both negation-as-failure and classical negation, relieved range-restrictedness in the rule heads, remote binding for cooperative inferencing among multiple rule engines.

(b)Flora is an advanced object-oriented knowledge representation and reasoning system. It is a dialect of F-logic with numerous extensions, including meta-programming in the style of HiLog, logical updates in the style of Transaction Logic, and defeasible reasoning. Applications include intelligent agents, Semantic Web, knowledge-based networking, ontology management, integration of information, security policy analysis, and more.

Here is the link:http://flora.sourceforge.net/

The Frame Logic (or F-logic) provides a logical foundation for frame-based and object-oriented languages for data and knowledge representation.

HiLog is a logical formalism that provides higher-order and meta-programming features in a computationally tractable first-order setting.

Transaction Logic provides logical foundations for state changes and side effects in a logic programming language. A significant portion of this theory is implemented in Flora-2. Applications of Transaction Logic include modeling and reasoning about workflows, planning, robotics, view maintenance in databases, and more.

5 Ontology Matching

Knowledge Base	Classes	Instances	Subsumption
Food1	FruitOrVeggie Fruit StoneFruit CitrusFruit Veggie LeafyGreen Plant-Based Edible	Veggie(broccoli) LeaffyGreen(kale) StoneFruit(peach) CitrusFruit(tangerine) CitrusFruit(orange) Fruit(strawberry) Fruit(apple) Plant-Based(soymilk) Plant-Based(oatmilk)	Fruit $\sqsubseteq$ FruitOrVeggie Veggie $\sqsubseteq$ FruitOrVeggie StoneFruit $\sqsubseteq$ Fruit CitrusFruit $\sqsubseteq$Fruit LeafyGreen $\sqsubseteq$ Veggy Plant-Based $\sqsubseteq$ Edible FruitOrVeggie $\sqsubseteq$ Edible
Food2	Dairy IceCream Plant Fruit RedFruit Vegetable Food	RedFruit(strawberry) RedFruit(raspberry) IceCream(strawberry) Vegetable(broccoli) Vegetable(lettuce) Dairy(milk) Fruit(tangelo) Fruit(orange) Plant(oat)	RedFruit $\sqsubseteq$ Fruit Fruit $\sqsubseteq$ Plant Vegetable $\sqsubseteq$ Plant Plant $\sqsubseteq$ Food Dairy $\sqsubseteq$ Food IceCream $\sqsubseteq$ Dairy

Figure 6 - two small food related KBs

a) Class labels comparison

In a), I use Levenshtein distance to calculate the edit distance and

similarity between all pairs of classes, in Food1 and Food2.

The result is as follow:

	Dairy	IceCream	Plant	Fruit	RedFruit	Vegetable	Food
FruitOrVeggie	11 0.15	11 0.15	12 0.08	8 0.38	11 0.15	11 0.15	12 0.08
Fruit	5 0.0	7 0.125	4 0.20	0 1.0	3 0.625	8 0.11	4 0.20
StoneFruit	9 0.10	8 0.20	8 0.20	5 0.5	5 0.5	9 0.10	9 0.10
CitrusFruit	10 0.10	10 0.10	10 0.10	6 0.45	6 0.45	11 0.0	10 0.10
Veggie	6	7	6	5	6	5	6

	0.0	0.125	0.0	0.17	0.25	0.44	0.0
LeafyGreen	8 0.20	8 0.20	9 0.10	9 0.10	8 0.20	8 0.20	10 0.0
Plant-Based	10 0.10	10 0.10	6 0.45	10 0.10	11 0.0	9 0.18	10 0.10
Edible	5 0.17	7 0.125	6 0.0	6 0.0	7 0.125	6 0.33	6 0.0

Chart 1 - the result of Levenshtein distance and similarity(The first line is the edit distance and the second is the similarity)

I define if they are matched when the result is above 0.4.

So the result is (Fruit,Fruit), (Fruit,RedFruit), (StoneFruit,Fruit), (StoneFruit,RedFruit),

(CitrusFruit,Fruit) ,(CitrusFruit,RedFruit), (Veggie,Vegetable), (Plant-Based,Plant).

The Java code is as follow:

//calculate levenshtein distance

The java code is as follow:

```
public static void levenshtein(String first,String second) {
    int len1 = first.length();
    int len2 = second.length();
    int[][] dif = new int[len1 + 1][len2 + 1];
    for (int a = 0; a <= len1; a++) {
       dif[a][0] = a;  }
    for (int a = 0; a <= len2; a++) {
       dif[0][a] = a;  }
    int temp;
    for (int i = 1; i <= len1; i++) {
       for (int j = 1; j <= len2; j++) {
          if (first.charAt(i - 1) == second.charAt(j - 1)) {
             temp = 0;  }
                          else {
             temp = 1;  }
          dif[i][j] = min(dif[i - 1][j - 1] + temp, dif[i][j - 1] + 1,
```

```
                dif[i - 1][j] + 1);  } }
    System.out.println("The comparison result of String\""+first+"\"and\""+second+"\"");
    System.out.println("Edit Distance："+dif[len1][len2]);
    float similarity =1 - (float) dif[len1][len2] / Math.max(first.length(), second.length());
    System.out.println("Similarity："+similarity);
  }
  private static int min(int... is) {
    int min = Integer.MAX_VALUE;
    for (int i : is) {
      if (min > i) {
        min = i;  } }
    return min; }
```

b) Class instances comparison.

First we reuse Levenshtein distance to calculate the edit distance and

similarity between all pairs of instances, in Food1 and Food2.

	strawberry	raspberry	broccoli	lettuce	milk	tangelo	orange	oat
broccoli	9 0.10	9 0.0	0 1.0	8 0.0	7 0.125	7 0.125	7 0.125	7 0.125
kale	8 0.20	7 0.22	7 0.125	6 0.14	3 0.25	5 0.29	4 0.33	3 0.25
peach	9 0.10	7 0.22	7 0.125	5 0.29	5 0.0	7 0.0	5 0.17	4 0.20
tangerine	7 0.30	7 0.22	9 0.0	8 0.11	8 0.11	4 0.56	16 0.33	8 0.11
orange	7 0.30	7 0.22	7 0.125	6 0.14	6 0.0	4 0.43	0 1.0	4 0.33
strawberry	0 1.00	4 0.60	9 0.10	9 0.10	10 0.0	7 0.3	7 0.3	9 0.10
apple	9 0.10	6 0.33	7 0.125	6 0.14	4 0.20	5 0.29	5 0.17	5 0.0

soymilk	9 0.10	8 0.11	6 0.25	7 0.0	3 0.57	6 0.14	6 0.14	6 0.14
oatmilk	9 0.10	8 0.11	7 0.125	6 0.14	3 0.57	5 0.29	6 0.14	4 0.43

Chart 2 - the result of Levenshtein distance and similarity of instances(The first line is the edit distance and the second is the similarity)

For all classes, I calculate how many instances they both have to calculate Jaccard similarity.

	Dairy	IceCream	Plant	Fruit	RedFruit	Vegetable	Food
FruitOrVeggie	0.125	0.14	0.27	0.22	0.125	0.125	0.25
Fruit	0.17	0.2	0.2	0.29	0.167	0	0.15
StoneFruit	0	0	0	0	0	0	0
CitrusFruit	0	0	0.125	0.2	0	0	0.11
Veggie	0	0	0.125	0	0	0.33	0.11
LeafyGreen	0	0	0	0	0	0	0
Plant-Based	0	0	0	0	0	0	0
Edible	0.1	0.11	0.23	0.18	0.1	0.1	0.21

Chart 3 - the result of Jaccard to calculate the similarity of classes.

Because the condition is very strict, so we think if the result is above 0.15, the two classes are matched.

So the result is (Fruit,Dairy), (Fruit,IceCream), (Fruit,Plant), (FruitOrVeggie,Plant), (Fruit,Fruit), (FruitOrVeggie,Fruit), (Fruit,RedFruit), (Fruit,Food), (FruitOrVeggie,Food), (CitrusFruit,Fruit), (Veggie,Vegetable), (Edible,Plant), (Edible,Fruit),(Edible,Food).

In this part, I calculated them by hand.

c) Taxonomy comparison

Then I use Upward co-topic distance to calculate the class similarity based on taxonomy.

	Dairy	IceCream	Plant	Fruit	RedFruit	Vegetable	Food
FruitOrVeggie	0	0	0	0	0	0	0
Fruit	0	0	0	0.2	0.17	0	0

StoneFruit	0	0	0	0.17	0.14	0	0
CitrusFruit	0	0	0	0.17	0.14	0	0
Veggie	0	0	0	0	0	0	0
LeafyGreen	0	0	0	0	0	0	0
Plant-Based	0	0	0	0	0	0	0
Edible	0	0	0	0	0	0	0

Chart 4 - the result of Upward co-topic distance to calculate the similarity of classes.

For every class, only when their parents' classes are the total same, they will be calculated into the results.

Because the condition is very strict, so we think if the result is above 0.15, the two classes are matched.

So the result is (Fruit,Fruit), (StoneFruit,Fruit), (CitrusFruit,Fruit), (Fruit,RedFruit).

The java code is as follow:

```
//Define the tree node.
public class TreeNode {
   private String nodeId;
   private String parentId;
   private String text;
   public TreeNode(String nodeId)
   {  this.nodeId = nodeId;  }
   public TreeNode(String nodeId, String parentId)
   { this.nodeId = nodeId;
     this.parentId = parentId;  }
   public String getNodeId() {
      return nodeId;  }
   public void setNodeId(String nodeId) {
      this.nodeId = nodeId; }
   public String getParentId() {
```

```
        return parentId;  }
    public void setParentId(String parentId) {
        this.parentId = parentId; }
    public String getText() {
        return text;  }
    public void setText(String text) {
        this.text = text;  } }

//Define Multi-fork tree node.
public class ManyTreeNode
{     private TreeNode data;
    private List<ManyTreeNode> childList;
    public ManyTreeNode(TreeNode data)
 {  this.data = data;
    this.childList = new ArrayList<ManyTreeNode>();  }

    public ManyTreeNode(TreeNode data, List<ManyTreeNode> childList)
    {  this.data = data;
       this.childList = childList;  }
    public TreeNode getData() {
        return data;  }
    public void setData(TreeNode data) {
      this.data = data;  }
    public List<ManyTreeNode> getChildList() {
        return childList;  }
    public void setChildList(List<ManyTreeNode> childList) {
        this.childList = childList;}}
//Define the tree
public class ManyNodeTree
{    private ManyTreeNode root;
            public ManyNodeTree()
            {root = new ManyTreeNode(new TreeNode("root"));}
            public ManyNodeTree createTree(List<TreeNode> treeNodes){
            if(treeNodes == null || treeNodes.size() < 0)
```

```
        return null;
        ManyNodeTree manyNodeTree =  new ManyNodeTree();
                for(TreeNode treeNode : treeNodes){
                if(treeNode.getParentId().equals("root")){
        manyNodeTree.getRoot().getChildList().add(new ManyTreeNode(treeNode));}
                else{addChild(manyNodeTree.getRoot(), treeNode);}}
                return manyNodeTree;}

        public void addChild(ManyTreeNode manyTreeNode, TreeNode child)
        {for(ManyTreeNode item : manyTreeNode.getChildList())
        {if(item.getData().getNodeId().equals(child.getParentId()))
        {item.getChildList().add(new ManyTreeNode(child));
        break;}
        else
        {if(item.getChildList() != null && item.getChildList().size() > 0)
        {addChild(item, child);}}}}
        public String iteratorTree(ManyTreeNode manyTreeNode)
        {       StringBuilder buffer = new StringBuilder();
                buffer.append("\n");
                if(manyTreeNode != null)
                {       for (ManyTreeNode index : manyTreeNode.getChildList()) {
                                buffer.append(index.getData().getNodeId()+ ",");

        if (index.getChildList() != null && index.getChildList().size() > 0 )
                {buffer.append(iteratorTree(index));}}}
                buffer.append("\n");
                return buffer.toString();}
public static TreeNode getLastCommonNode(List<TreeNode> p1, List<TreeNode> p2) {
        Iterator<TreeNode> ite1 = p1.iterator();
    Iterator<TreeNode> ite2 = p2.iterator();
     TreeNode last = null;
     while (ite1.hasNext() && ite2.hasNext()) {
       TreeNode tmp = ite1.next();
       if (tmp == ite2.next()) {
```

```
        last = tmp;}}
    return last;}
public static void getNodePath(TreeNode root, TreeNode target, List<TreeNode> path) {
    if (root == null) {return;}
    path.add(root);
    List<TreeNode> children = root.children;
    for (TreeNode node : children) {
        if (node == target) {
            path.add(node);
            return;
        } else {
            getNodePath(node, target, path);}}
    path.remove(path.size() - 1);}
        public static TreeNode getLastCommonParent(TreeNode root, TreeNode p1, TreeNode p2) {
    if (root == null || p1 == null || p2 == null) {
        return null;}
    List<TreeNode> path1 = new LinkedList<>();
    getNodePath(root, p1, path1);
    List<TreeNode> path2 = new LinkedList<>();
    getNodePath(root, p2, path2);
    return getLastCommonNode(path1, path2);}
        public ManyTreeNode getRoot() {return root;}
        public void setRoot(ManyTreeNode root) {
        this.root = root;
        public static void main(String[] args)
        {                       List<TreeNode> treeNodes1 = new ArrayList<TreeNode>();
                                treeNodes1.add(new TreeNode("FruitOrVeggie", "Edible"));
                                treeNodes1.add(new TreeNode("Fruit", "FruitOrVeggie"));
                                treeNodes1.add(new TreeNode("CitrusFruit", "Fruit"));
                                treeNodes1.add(new TreeNode("StoneFruit", "Fruit"));
                                treeNodes1.add(new TreeNode("LeafyGreen", "Veggie"));
                                treeNodes1.add(new TreeNode("Veggie", "FruitOrVeggie"));
                                treeNodes1.add(new TreeNode("Plant-Based", "Edible"));
                                List<TreeNode> treeNodes2 = new ArrayList<TreeNode>();
```

```
treeNodes2.add(new TreeNode("Plant", "Food"));
treeNodes2.add(new TreeNode("Dairy", "Food"));
treeNodes2.add(new TreeNode("Fruit", "Plant"));
treeNodes2.add(new TreeNode("Vegetable", "Plant"));
treeNodes2.add(new TreeNode("RedFruit", "Fruit"));
treeNodes2.add(new TreeNode("IceCream", "Dairy"));
ManyNodeTree tree = new ManyNodeTree();
//than get their common ancestor one by one.}}
```

d) Merging strategy

We choose the max similarity results in the 3 methods.

	Dairy	IceCream	Plant	Fruit	RedFruit	Vegetable	Food
FruitOrVeggie	0.15	0.15	0.27	0.38	0.15	0.15	0.25
Fruit	0.17	0.2	0.2	1.0	0.625	0.11	0.20
StoneFruit	0.17	0.20	0.2	0.5	0.5	0.1	0.1
CitrusFruit	0.1	0.1	0.125	0.45	0.45	0	0.11
Veggie	0	0.125	0.125	0.17	0.25	0.44	0.11
LeafyGreen	0.20	0.2	0.1	0.10	0.20	0.20	0
Plant-Based	0.10	0.1	0.45	0.10	0.0	0.18	0.10
Edible	0.17	0.125	0.23	0.18	0.125	0.33	0.21

Chart 5 - the result of Merging strategy to add up the similarity of classes.

In this part, we define that if the result is above 0.4, we think they are matched.

So the result is (Fruit,Fruit), (Fruit,RedFruit), (StoneFruit,Fruit), (StoneFruit,RedFruit),

(CitrusFruit,Fruit) ,(CitrusFruit,RedFruit), (Veggie,Vegetable), (Plant-Based,Plant).

The java code is as follow:

```
int Choosemax(int a,int b,int c)
{       int t;
        if(a>b)t=a;
```

```
else t=b;
if(c>t)t=c;
return t;}
```

e) Discussion

By using three different comparison methods, I think using Levenshtein distance to calculate the similarity is more reliable. Because the two knowledge base is very different from each other, including classes, instances and subsumptions. Taxonomy comparison is Problematic. If food1 and food2 have more exact the same instances, then Jaccard (instance-based) can be the best choice to calculate the similarity of classes. Also, if food1 and food2 have more exact the same classes, taxonomy comparison is more suitable to calculate the class similarity.

6 References.

[1] CSI 5180 Topics in AI - Ontologies and Semantic Web -Assignment 2 - Reasoning and Ontology Matching. Retrieved from https://uottawa.brightspace.com/d2l/le/content/102243/viewContent/2195996/View

[2] Flora-2 official website. Retrieved from http://flora.sourceforge.net/

[3] Bossam: An Extended Rule Engine for OWL Inferencing. Retrieved from https://www.bvb.de/eng