BIG DATA

Web semantic

22/03/2019

RDF Graph saturation

By :

Sofien Omri

Taieb Jlassi

Hamza Ferchichi

Under supervision of

Prof. Khalid Belhajjem

Academic year: 2018/2019

Report abstract

This report describes the process used to implement RDF saturation using PySpark in the Google Colab plateform. In the first section, we present a brief overview of the inference rules that saturate RDF graphs. Then the second section presents the technique used to implement the RDF rules number 2, 3, 7 and 9 using Spark and finally we have Test the above rules using example of NTRIPLE files.

List of figures

List of tables

Contents

1. Introduction

Saturation of the database with new data (metadata or links between contents) will make it possible to enrich the partners' corpuses and improve the quality of research results. Since in the context of the semantic web the inference rules allowing deducing new triplets, the saturation operation of an RDF graph consists in applying these inference rules in order to materialize all the deductible triplets. This results in a saturated graph.

Therefore, this report describes the process used to implement RDF saturation using Spark.

In the first section, we present a brief overview of the inference rules that saturate RDF graphs. Then the second section presents the technique used to implement the RDF rules number 2, 3, 7 and 9 using Spark and finally we have Test the above rules using example of NTRIPLE files.

2. Basic concepts

In this section, we first describe the RDF data model, the RDF and OWL vocabularies. Next, we present the list of rules proposed by the W3C.

2.1. The RDF data model

RDF is an assertion language designed to express proposals using precise formal vocabularies, including those defined with RDFS (RDF Schema) or by OWL (Conen et al. 2000). RDF is a W3C standard that is based on graphs. An RDF graph is a set of triplets (s, p, and o) with s is the subject having as property p, and the value of this property is the object o.

The sets of triplets build an oriented graph whose nodes and URIs (uniform resource identifiers) labels arcs. RDF includes a set of classes and properties that have namespaces beginning with rdf: or rdfs: For example rdf: type specifies the type of class to which the resource belongs.

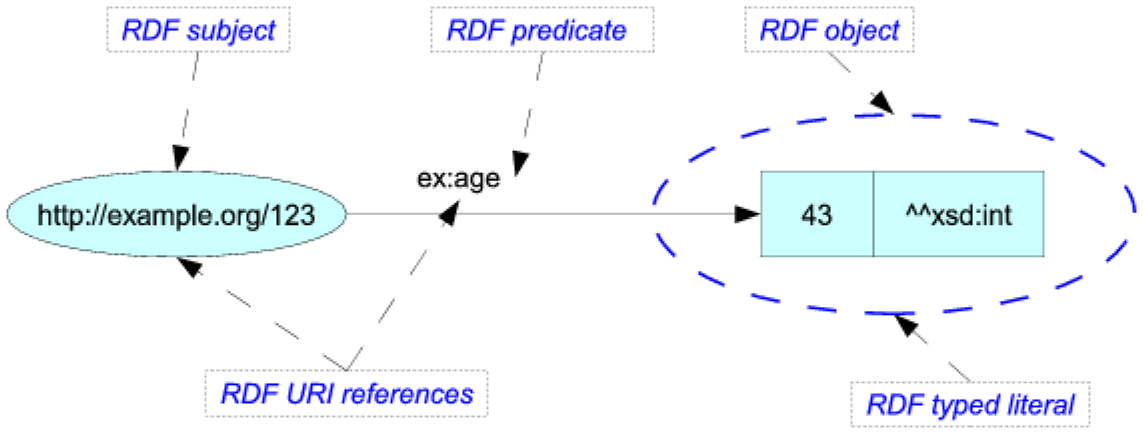


Figure 1: RDF graph example

2.2. RDFS

RDFS is part of the RDF recommendation for the description of its vocabulary. It has been designed to put constraints between classes and properties in an RDF graph. RDFS allows you to declare, in RDF, the properties and classes used to describe RDF data and the hierarchical relationships that exist between properties and between classes. It allows you to define:

- A resource as a class of type rdfs: Class. A class is a resource representing a set of resources whose rdf: type property value is this class ;
- A resource as being a property of type rdf: Property ;
- Class hierarchies with the property rdfs:subClassOf ;
- Property hierarchies with the property rdfs:subPropertyOf ;
- The domain and scope of a property with the properties rdfs:domain and rdfs:range ;
- The labels of classes and properties with the property rdfs:label ;
- Natural language definitions of classes and properties with the property rdfs: how.

2.3. OWL

However, to define rich ontologies, the expressiveness of RDFS becomes insufficient. Indeed, RDFS does not allow expressing uniqueness, union, class intersection, nor some algebraic properties of properties such as transitivity, symmetry, etc. This is what motivated the recommendation of Web Ontology Language (OWL), which allows not only to declare properties and classes but also to define them. OWL was recommended by the W3C in 2004 and its version 2 in 2009.

- It is a meta-vocabulary (like RDFS) inspired by the logic of descriptions with concrete (literal) values.

- It defines several profiles offering different compromises in terms of expressiveness and complexity.

- It mimics the meta-modeling capabilities of RDFS.

- The list of the main elements that can be expressed by OWL:

- Equivalence of classes, using the owl: equivalentClass property,

- Equivalence of properties, using the owl: equivalentProperty property,

- Reverse relationships, using the owl: inverseOf property,

- Symmetric relationships, using the owl: SymmetricProperty class,

- Transitive relationships, using the owl: TransitiveProperty class,

- Property restrictions, using the owl class: Restriction and property owl: onProperty,

- Value constraints, using the properties owl: allValuesFrom, owl: someValuesFrom, owl: hasValue.

2.4. RDF Inference Rule (entailment rules)

More generally, an inference rule is a function that takes tuple of formulas and returns a formula. In the semantic web domain, inference is an action or process that allows you to discover new expressions. Implied triplets are considered in the graph.

An important feature of RDF is the modeling of implicit triplets: they are considered part of a graph, even if they are not explicitly present in it. The W3C names RDF denote the mechanism by which implicit triplets are derived (or generated) from the explicit triplets of a graph and notch rules (or also inference rules).

In RDF, there are four notch regimes (ter et al., 2005)

- Simple nick: it does not interpret the RDF and RDFS vocabulary

- RDF notch: interpreted the RDF vocabulary

- RDFS notch: interpreted the RDF and RDFS vocabularies

- Entailment- D: additionally uses information on data types

In our project, we just use the RDFS rules and in the following we will present these techniques and we just focus only on the rules number two, three, seven and nine (see the table below).

2.5. RDFS entailment

RDFS inferences satisfy the lemma: a set of graphs G of type RDF implies a graph E if only if there is a graph that can be derived from G and the axiomatic triplets of RDF and RDFS by applying the rules RDFS and that simply implies E. these rules can be presented in table below.

Table 1: RDFS entailment rules

Rule	If E contains	Add
Rdfs2	aaa rdfs: domain xxx. yyy aaa zzz	yyy rdf :type xxx
Rdfs3	aaa rdfs :range xxx. yyy aaa zzz	zzz rdf :type xxx
Rdfs7	aaa rdfs :subpropertyof bbb xxx aaa yyy	xxx bbb yyy
Rdfs9	xxx rdfs:subclassof yyy zzz rdf:type xxx	zzz rdf:type yyy

3. Saturation algorithm

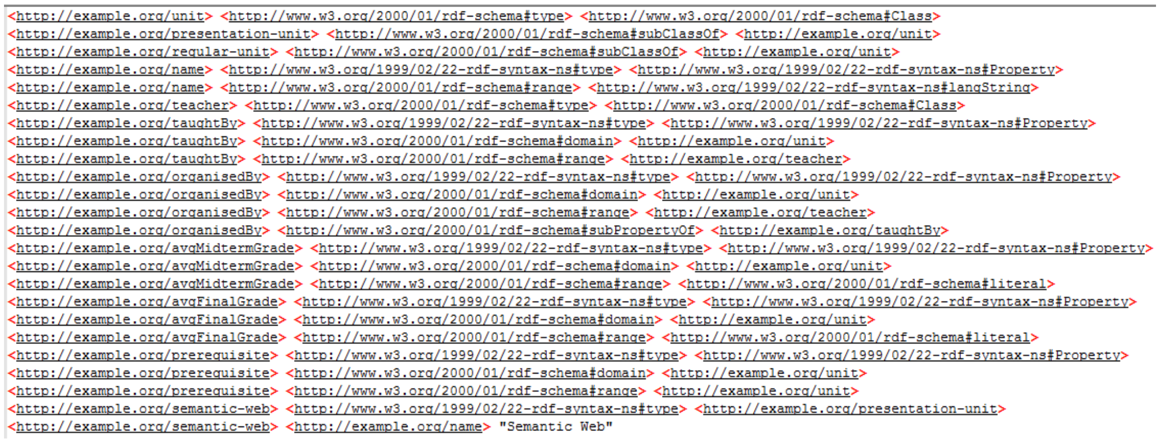
Evaluating a query q in an RDF G graph using only the explicit queries in the graph usually gives incomplete information.

For this reason, we will use the graph saturation technique. As defined in (Bursztyn et al.,2015), Immediate inference rules allow to define a finite saturation of an RDF G graph which is a G∞ graph defined as a fixed point, by applying a set of an inference rules. The saturation of an RDF graph is unique and no longer contains implicit triplets. It is important to note that the RDF notch is part of the RDF standard. Therefore, in RDF, any graph G is (semantically) equivalent to its saturation G∞.

Table 2: Data description

General overview	**Title**	Handin.nt
	Creator	• Søren Kejser Jensen • Johannes L. Borresen
	Date	Oct 7, 2016
	Method	The data was created by using the SPARQL query[1]
	Source	GitHub[2]
Content description	**Subject**	This data is about the structure of school classes. It contains course subjects as web semantic, xml and Internet programing. We also find the teachers' name.
Technical description	**File formats**	This data is an RDF graph, which is represented by the N-triples format (.nt).
	Necessary Software	This data was saturated by the PySpark software in the Google Colab plateform

[1] You can find the SPARQL code in this link : https://github.com/Tellus/neo4j-rdf/blob/master/data/query.sqf .
[2] https://github.com/Tellus/neo4j-rdf/tree/master/data .



```
<http://example.org/unit> <http://www.w3.org/2000/01/rdf-schema#type> <http://www.w3.org/2000/01/rdf-schema#Class>
<http://example.org/presentation-unit> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://example.org/unit>
<http://example.org/regular-unit> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://example.org/unit>
<http://example.org/name> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://example.org/name> <http://www.w3.org/2000/01/rdf-schema#range> <http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
<http://example.org/teacher> <http://www.w3.org/2000/01/rdf-schema#type> <http://www.w3.org/2000/01/rdf-schema#Class>
<http://example.org/taughtBy> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://example.org/taughtBy> <http://www.w3.org/2000/01/rdf-schema#domain> <http://example.org/unit>
<http://example.org/taughtBy> <http://www.w3.org/2000/01/rdf-schema#range> <http://example.org/teacher>
<http://example.org/organisedBy> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://example.org/organisedBy> <http://www.w3.org/2000/01/rdf-schema#domain> <http://example.org/unit>
<http://example.org/organisedBy> <http://www.w3.org/2000/01/rdf-schema#range> <http://example.org/teacher>
<http://example.org/organisedBy> <http://www.w3.org/2000/01/rdf-schema#subPropertyOf> <http://example.org/taughtBy>
<http://example.org/avgMidtermGrade> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://example.org/avgMidtermGrade> <http://www.w3.org/2000/01/rdf-schema#domain> <http://example.org/unit>
<http://example.org/avgMidtermGrade> <http://www.w3.org/2000/01/rdf-schema#range> <http://www.w3.org/2000/01/rdf-schema#literal>
<http://example.org/avgFinalGrade> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://example.org/avgFinalGrade> <http://www.w3.org/2000/01/rdf-schema#domain> <http://example.org/unit>
<http://example.org/avgFinalGrade> <http://www.w3.org/2000/01/rdf-schema#range> <http://www.w3.org/2000/01/rdf-schema#literal>
<http://example.org/prerequisite> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://example.org/prerequisite> <http://www.w3.org/2000/01/rdf-schema#domain> <http://example.org/unit>
<http://example.org/prerequisite> <http://www.w3.org/2000/01/rdf-schema#range> <http://example.org/unit>
<http://example.org/semantic-web> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/presentation-unit>
<http://example.org/semantic-web> <http://example.org/name> "Semantic Web"
```

Figure 2: Sample of the data

4. The implementation of the RDFS rules

In order to simplify and facilitate our work, we have implemented these rules by using RDFs of the N-triples form which designed to be a simpler format than Notation 3 and Turtle, and therefore easier for software to parse and generate. To do that, we just use the spark software in the Google Colab platform.

Code Description:

We can find the code of each RDFS rules implemented in the appendix or the Google Colab shared file with have make some comment to each code:

Rules 2: rdfs: Domain:

To implement this rule we tried to adapt it with a small number of RDFs so that we can then use them on a large-scale database.

We can take for example the two RDF triple shown below:

```
'<http://example.org/taughtBy> <http://www.w3.org/2000/01/rdf-schema#domain> <http://example.org/unit>',
```

```
'<http://example.org/xml> <http://example.org/taughtBy> <http://example.org/bob>',
```

The spark output:

After the application of the RDFs rule number two we have reached this result and you can find the code attach in the Google Colab shared file:

```
'<http://example.org/xml> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/unit>'])
```

Rule3: rdfs: range

The input:

```
'<http://example.org/organisedBy> <http://www.w3.org/2000/01/rdf-schema#range> <http://example.org/teacher>',
```

```
 '<http://example.org/semantic-web> <http://example.org/organisedBy> <http://example.org/alice>',
```

The spark output:

After the application of the rule number 3, we have reached this result:

```
['<http://example.org/alice> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/teacher>',
```

Rule7: rdfs: SubpropertyOf

The input:

```
<http://example.org/organisedBy> <http://www.w3.org/2000/01/rdf-schema#subPropertyOf> <http://example.org/taughtBy>'
```

```
'<http://example.org/semantic-web> <http://example.org/organisedBy> <http://example.org/alice>'
```

```
<http://example.org/xml> <http://example.org/organisedBy> <http://example.org/carole>'
```

The spark output:

```
 ['<http://example.org/semantic-web> <http://example.org/taughtBy> <http://example.org/alice>',
  '<http://example.org/xml> <http://example.org/taughtBy> <http://example.org/carole>']),
```

Rule9: rdfs: SubclassOf

The input:

```
'<http://example.org/presentation-unit> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://example.org/unit>',
```

```
'<http://example.org/semantic-web> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/presentation-unit>',
```

The spark output:

```
 ['<http://example.org/semantic-web> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/unit>']),
```

5. Conclusion:

In this report, we have managed to implement the four-entailment rules 2, 3, 7 and 9 of the rdfs entailment to saturate the RDF graph database. To do that, we just take as an example of an RDF schema with N-triples format and we have found that new RDF graph can be derived from this RDF graph and the axiomatic triplets of RDF and RDFS by applying the rules RDFS that make the RDF graph saturate and more explicit.

We tried to implement these four rules on the entire RDF a sample of instance triples from DBpedia with the entire RDF Schema triples of DBpedia but we found many of the problems that are due to the absence of conditions for the four rules to be applied.

6. References:

Bursztyn, Damian, et al. "Reasoning on Web Data: Algorithms and Performance." *ICDE-31st International Conference on Data Engineering*. 2015.

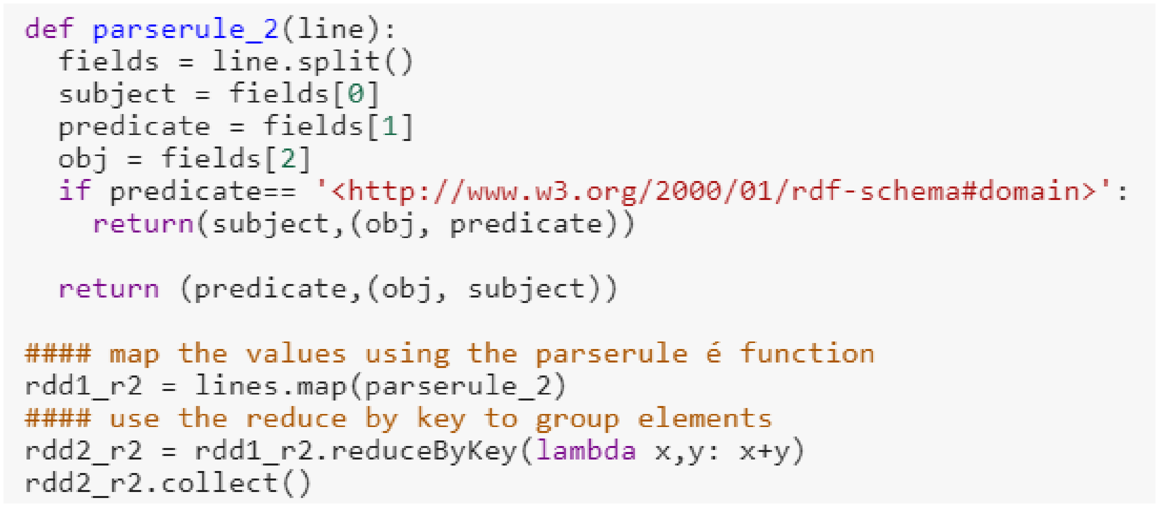
Calvanese, D., Giacomo, G. D., D. Lembo, M. Lenzerini, and R. Rosati, “Tractable Reasoning and Efficient Query Answering in escription Logics: The DL-Lite Family,” J. of Automated Reasoning (JAR), 2007.

Conen, Wolfram, and Reinhold Klapsing. "A logical interpretation of RDF."*Journal of Electronic Transactions on Artificial Intelligence (ETAI), Area: The Semantic Web (SEWEB)* 5 (2000).

https://github.com/Tellus/neo4j-rdf/tree/master/data

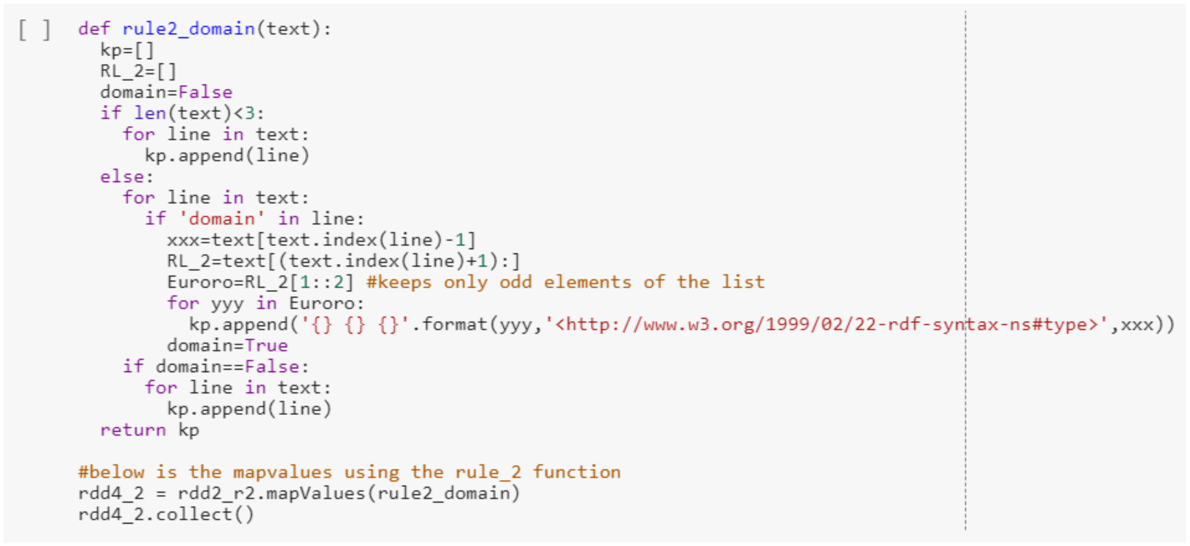
https://www.w3.org/TR/rdf-testcases/#ntriples

7. Appendix: Implementation Code



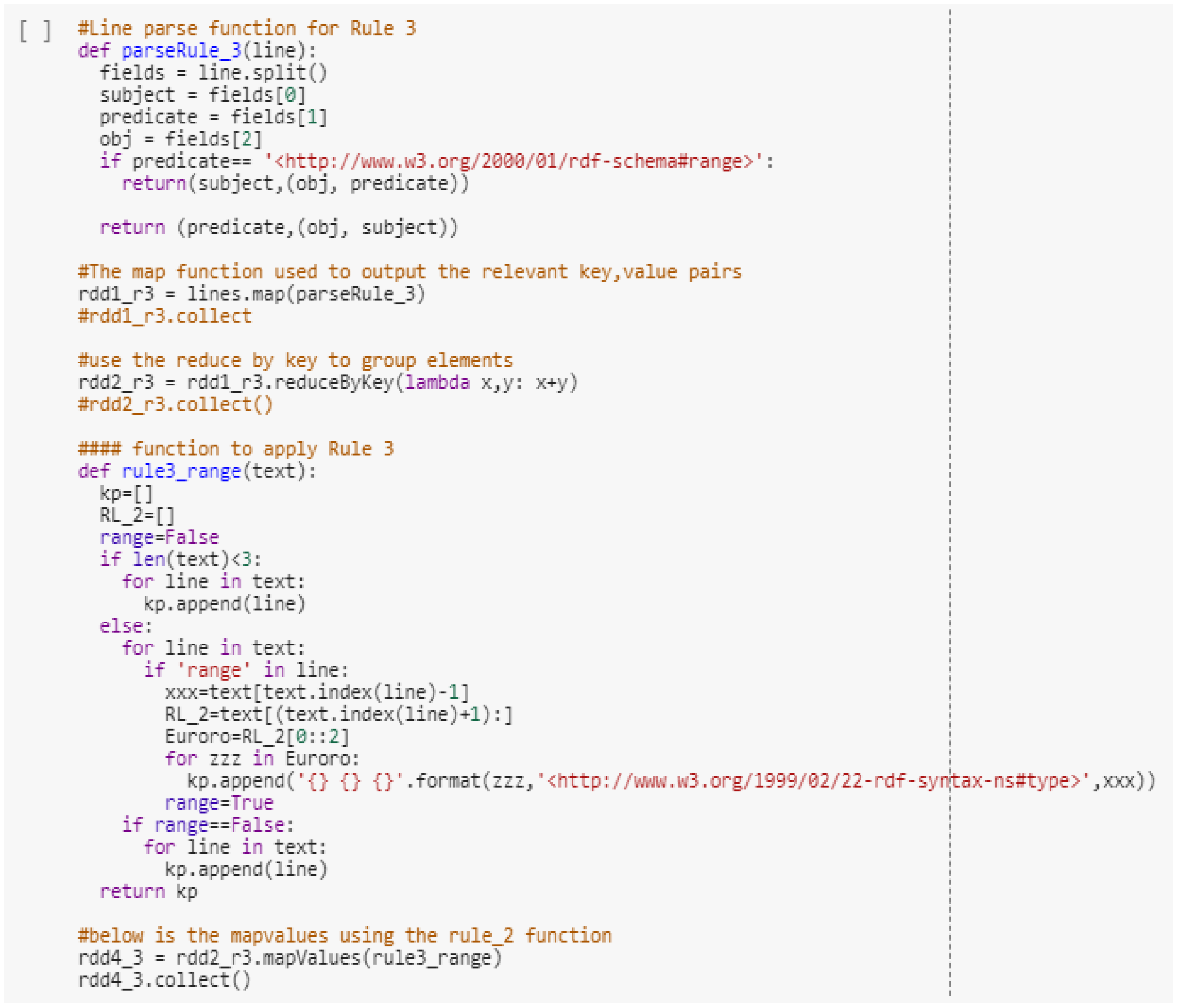
```
def parserule_2(line):
  fields = line.split()
  subject = fields[0]
  predicate = fields[1]
  obj = fields[2]
  if predicate== '<http://www.w3.org/2000/01/rdf-schema#domain>':
    return(subject,(obj, predicate))

  return (predicate,(obj, subject))

#### map the values using the parserule é function
rdd1_r2 = lines.map(parserule_2)
#### use the reduce by key to group elements
rdd2_r2 = rdd1_r2.reduceByKey(lambda x,y: x+y)
rdd2_r2.collect()
```

Figure 3: Map and Reduce transformations



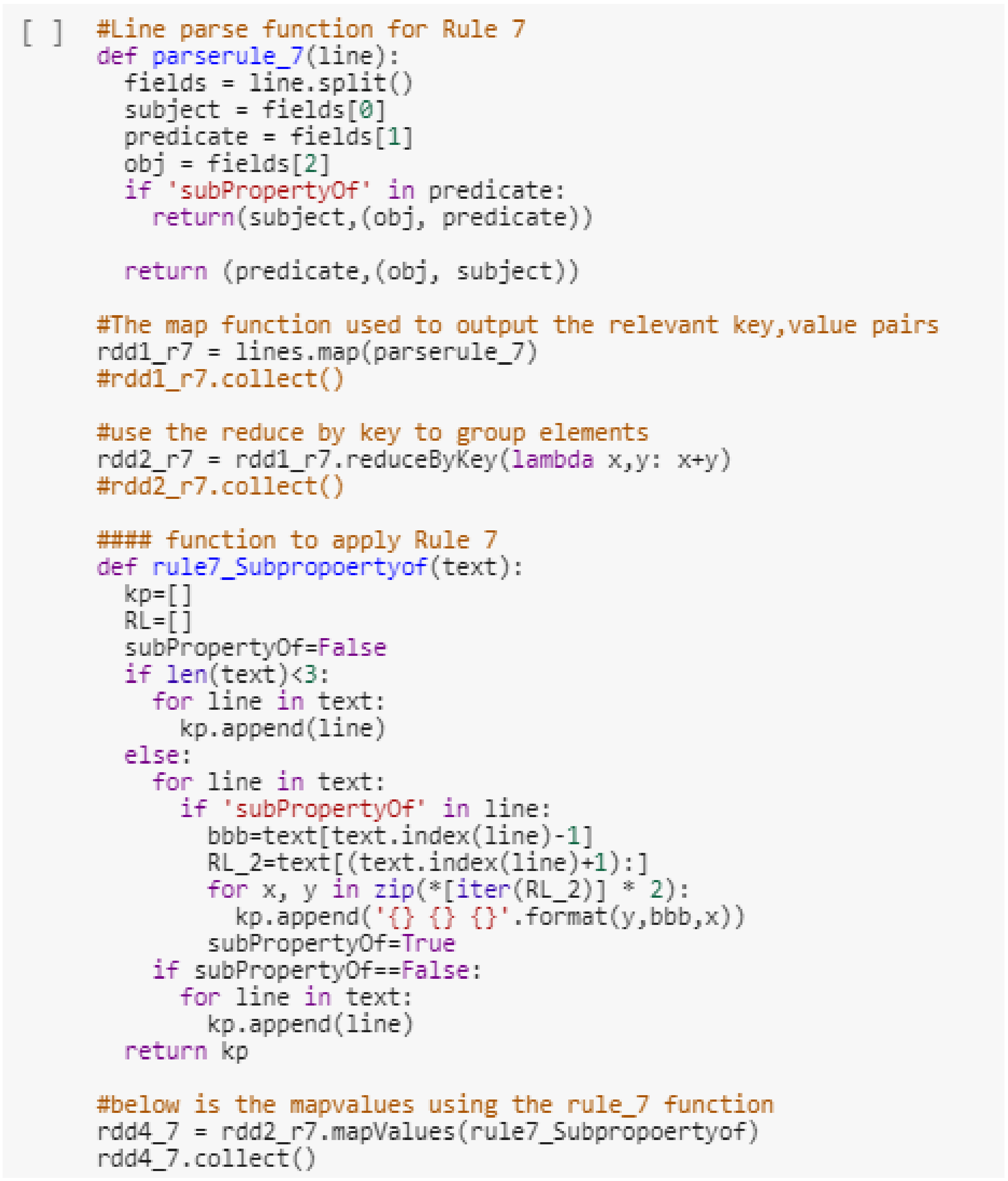
```
[ ]  def rule2_domain(text):
       kp=[]
       RL_2=[]
       domain=False
       if len(text)<3:
         for line in text:
           kp.append(line)
       else:
         for line in text:
           if 'domain' in line:
             xxx=text[text.index(line)-1]
             RL_2=text[(text.index(line)+1):]
             Euroro=RL_2[1::2] #keeps only odd elements of the list
             for yyy in Euroro:
               kp.append('{} {} {}'.format(yyy,'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',xxx))
             domain=True
         if domain==False:
           for line in text:
             kp.append(line)
       return kp

     #below is the mapvalues using the rule_2 function
     rdd4_2 = rdd2_r2.mapValues(rule2_domain)
     rdd4_2.collect()
```

Figure 4: Rule2: rdfs: Domain



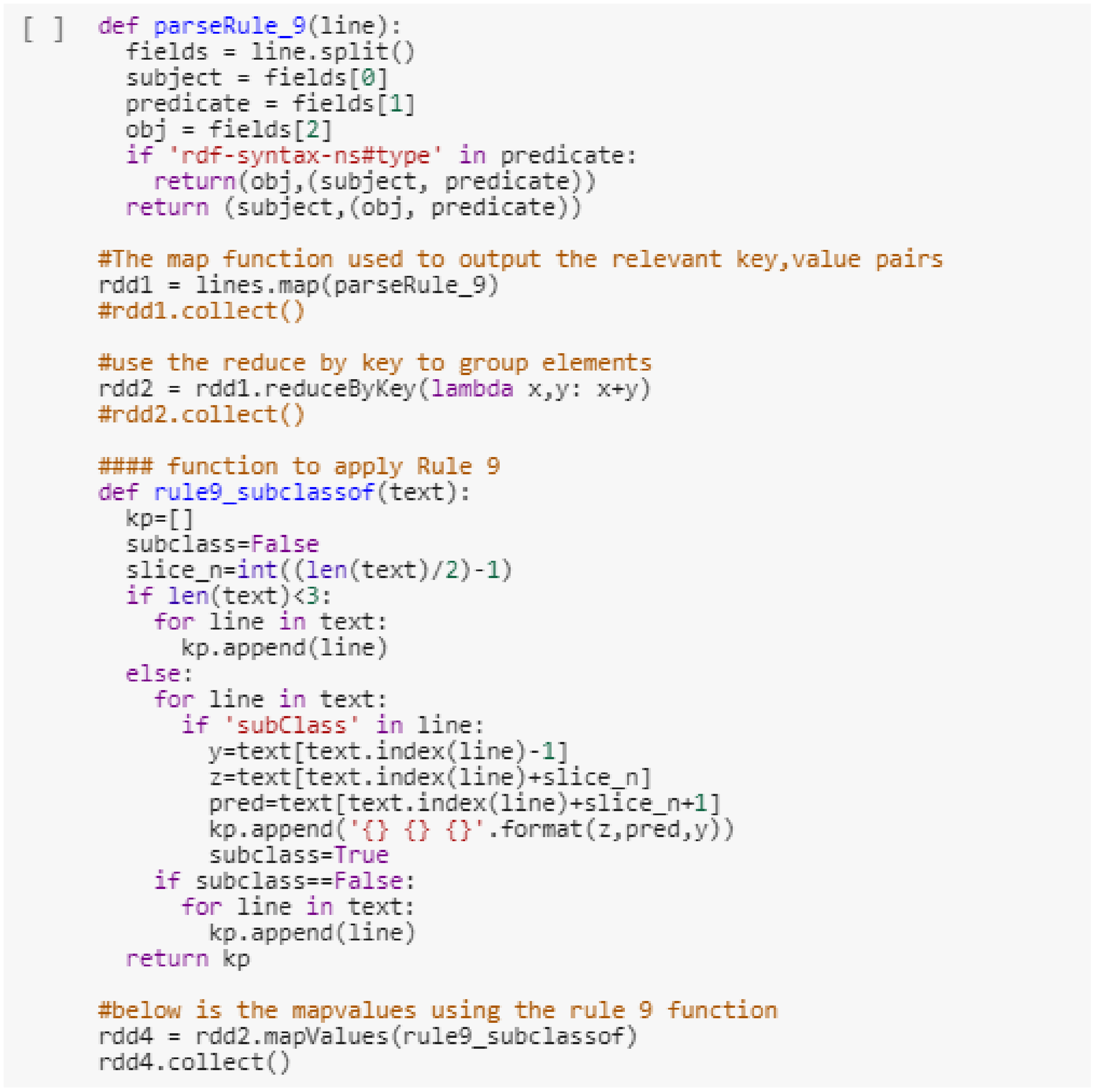
```
[ ]  #Line parse function for Rule 3
     def parseRule_3(line):
       fields = line.split()
       subject = fields[0]
       predicate = fields[1]
       obj = fields[2]
       if predicate== '<http://www.w3.org/2000/01/rdf-schema#range>':
         return(subject,(obj, predicate))

       return (predicate,(obj, subject))

     #The map function used to output the relevant key,value pairs
     rdd1_r3 = lines.map(parseRule_3)
     #rdd1_r3.collect

     #use the reduce by key to group elements
     rdd2_r3 = rdd1_r3.reduceByKey(lambda x,y: x+y)
     #rdd2_r3.collect()

     #### function to apply Rule 3
     def rule3_range(text):
       kp=[]
       RL_2=[]
       range=False
       if len(text)<3:
         for line in text:
           kp.append(line)
       else:
         for line in text:
           if 'range' in line:
             xxx=text[text.index(line)-1]
             RL_2=text[(text.index(line)+1):]
             Euroro=RL_2[0::2]
             for zzz in Euroro:
               kp.append('{} {} {}'.format(zzz,'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',xxx))
             range=True
         if range==False:
           for line in text:
             kp.append(line)
       return kp

     #below is the mapvalues using the rule_2 function
     rdd4_3 = rdd2_r3.mapValues(rule3_range)
     rdd4_3.collect()
```

Figure 5: Rule3: rdfs: Range



```
[ ]  #Line parse function for Rule 7
     def parserule_7(line):
       fields = line.split()
       subject = fields[0]
       predicate = fields[1]
       obj = fields[2]
       if 'subPropertyOf' in predicate:
         return(subject,(obj, predicate))

       return (predicate,(obj, subject))

     #The map function used to output the relevant key,value pairs
     rdd1_r7 = lines.map(parserule_7)
     #rdd1_r7.collect()

     #use the reduce by key to group elements
     rdd2_r7 = rdd1_r7.reduceByKey(lambda x,y: x+y)
     #rdd2_r7.collect()

     #### function to apply Rule 7
     def rule7_Subpropoertyof(text):
       kp=[]
       RL=[]
       subPropertyOf=False
       if len(text)<3:
         for line in text:
           kp.append(line)
       else:
         for line in text:
           if 'subPropertyOf' in line:
             bbb=text[text.index(line)-1]
             RL_2=text[(text.index(line)+1):]
             for x, y in zip(*[iter(RL_2)] * 2):
               kp.append('{} {} {}'.format(y,bbb,x))
             subPropertyOf=True
         if subPropertyOf==False:
           for line in text:
             kp.append(line)
       return kp

     #below is the mapvalues using the rule_7 function
     rdd4_7 = rdd2_r7.mapValues(rule7_Subpropoertyof)
     rdd4_7.collect()
```

Figure 6: Rule7: rdfs: SubpropertyOf



```
[ ]   def parseRule_9(line):
        fields = line.split()
        subject = fields[0]
        predicate = fields[1]
        obj = fields[2]
        if 'rdf-syntax-ns#type' in predicate:
          return(obj,(subject, predicate))
        return (subject,(obj, predicate))

      #The map function used to output the relevant key,value pairs
      rdd1 = lines.map(parseRule_9)
      #rdd1.collect()

      #use the reduce by key to group elements
      rdd2 = rdd1.reduceByKey(lambda x,y: x+y)
      #rdd2.collect()

      #### function to apply Rule 9
      def rule9_subclassof(text):
        kp=[]
        subclass=False
        slice_n=int((len(text)/2)-1)
        if len(text)<3:
          for line in text:
            kp.append(line)
        else:
          for line in text:
            if 'subClass' in line:
              y=text[text.index(line)-1]
              z=text[text.index(line)+slice_n]
              pred=text[text.index(line)+slice_n+1]
              kp.append('{} {} {}'.format(z,pred,y))
              subclass=True
          if subclass==False:
            for line in text:
              kp.append(line)
        return kp

      #below is the mapvalues using the rule 9 function
      rdd4 = rdd2.mapValues(rule9_subclassof)
      rdd4.collect()
```

Figure 7: Rule9: rdfs: SubclassOf