SDM Lab3: Knowledge-Graph

Davide Rendina
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
davide.rendina@estudiantat.upc.edu

Yi Wu
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
yi.wu@estudiantat.upc.edu

1 B.1 Creating TBox (programmatically)

Following the publication property graph modelled in the first lab of this course and using the same datasets (ref:Lab1_HernandezRendina) we built our TBOX using Python RDFLib [2022]. A visual representation of the TBOX was drawn using grafo and can be seen in fig.1. An image will also be attached to the deliverable for better readability.

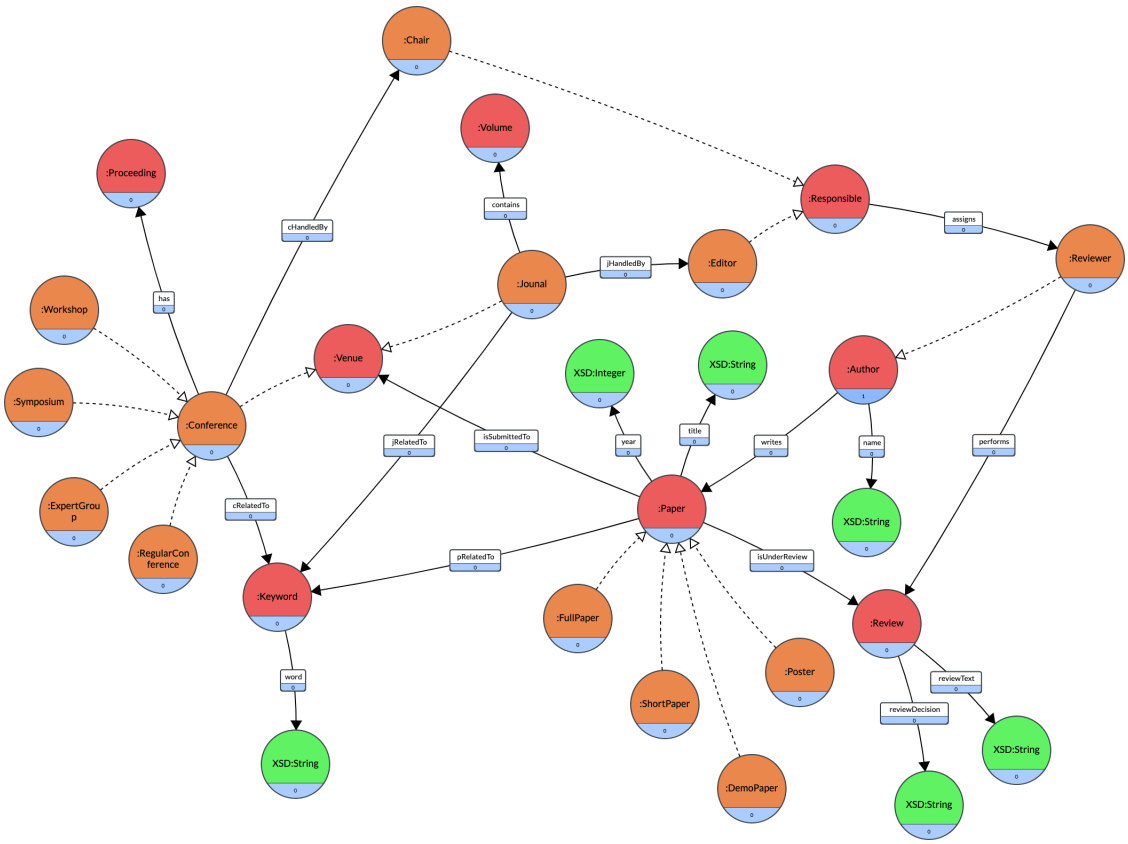


Figure 1: Publication data TBOX

In the graph we used different colors for marking superclasses (red), subclasses (orange) and XML Schema Data Types (green), which were used to represent attributes of the main

concepts. Where not specified in the lab description, the following assumptions had been made to model the ontology:

- Reviewer is a subclass of Author, we assume that reviewers are authors.
- The class Area, to which Conference, Journal and Paper are related, is represented by a Keyword (ref Lab1).
- Chair and Editor are a subclass of Responsible, which assigns the reviewers. Chair then manages Conference and Editor manages Journal.

The knowledge graph language of choice is RDFS, which would allow us to describe the different subclasses. The Turtle file containing the TBOX created programmatically using Python can be found attached to this submission in the Notebook B1_RendinaWu.

2 B.2 Creating ABox (programmatically)

The ABox was built using Python's library RDFLib, as per the TBox documented in the previous section.
The ABox was built in 3 steps: (1) generating the csv files containing the data, (2) Converting the csv into rdf, (3) output the ABox into file.

2.1 ABox Step 1 - Generating CSV files

The data was obtained for the csv files used int the SDM Lab 1 (ref Lab1_HernandezRendina). However, concepts like conference sub-classes, paper sub-classes, chair, editor, responsible, reviewer are synthetic.

Two CSVs will be joined first if necessary, to get the column values from two tables simultaneously.

2.2 ABox Step 2 - Converting the CSV data into RDF

Value parsing function 'URLparse' is built first, to standardize the data format, for example, changing space to underscore to fulfill the requirements of RDFLib.

```
def URLparse(url:str):
    url=url.replace("\'","_").replace("\"","_")
    for i in string.punctuation:
        url = url.replace(i,"_")
    url = url.replace(" ","_")
    return url
```

Instances are built into the graph while iterating each tuple of the CSV. The cell values of the target columns are added into the suffix of URI if it's an instance of a class. If it's a *XSD:String*, function *RDFLib.Literal* will be called.

```
df_authors=pd.read_csv(directory+"/ABox_data/authors.csv")
for index,row in df_authors.iterrows():
    authors_author_id = row['author_id']
    authors_author_name = Literal(str(row['author_name']))
    # create ABox
    author_node = URIRef(f"http://sdm_lab/{URLparse(authors_author_id)}")
    g.add((author_node, RDF.type, sdm_lab.Author))
    g.add((author_node, sdm_lab.name, authors_author_name))
df_authors.head(2)
```

In the example code above, authors_node instance is defined explicitly by *g.add((author_node, RDF.type, sdm_lab.Author))*.

However, thanks to the RDFS inference mechanism, we don't have to claim author instance is a RDF.type of class Author and paper instance is a RDF.type of class Paper in the relationship

'author writes paper', since it can be inferred as the domain of the property 'writes' defined in the TBox. Such inferences are assigned a context of 'http://www.ontotext.com/implicit' in graphDB, and can be queried through SPARQL.
Therefore, the relationship 'author writes paper' is converted saving defining the two RDF.type for the classes, as follows:

```
df_author_wrote_paper=pd.read_csv(directory+"/ABox_data/author_writes_paper.csv")
for index,row in df_author_wrote_paper.iterrows():
    author_wrote_paper_author_id = row['author_id']
    author_wrote_paper_paper_id = row['paper_id']
    # create ABox
    author_node = URIRef(f"http://sdm_lab/{URLparse(author_wrote_paper_author_id)}")
    paper_node = URIRef(f"http://sdm_lab/{URLparse(author_wrote_paper_paper_id)}")
    g.add((author_node, sdm_lab.writes, paper_node))
```

Same applies for *g.add((s,p,o))* structure works the same, with no need to claim that the instance is a RDF.type as this would be implicitly inferred by the domain or range.

2.3 ABox Step 3 - Ouput Abox

Finally, the ABox file is output by function *graph.serialize* in format *.nt*.

3 B.3 Creating the final ontology

Once generated the TBox and Abox, the two files were imported into our GraphDB repository. The ruleset of our repository was set to RDFS (Optimised), see Fig 2 which according to GraphDB [2022] "supports the standard model-theoretic RDFS semantics. This includes support for subClassOf and related type inference, as well as subPropertyOf". In the figure below we can see our class hierarchy after loading the two files.

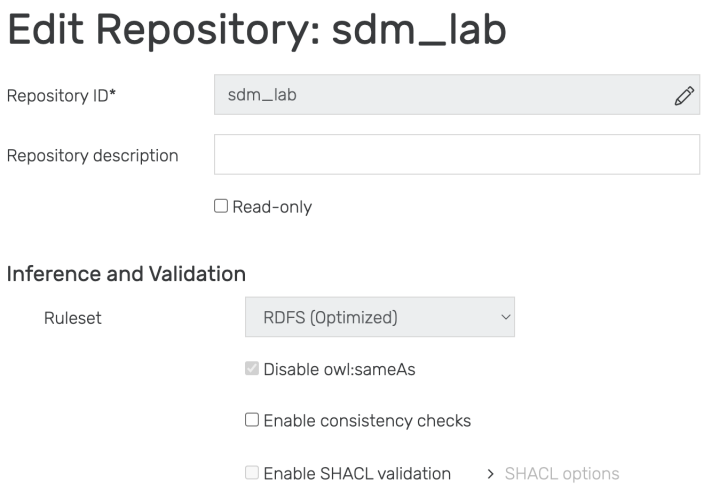


Figure 2: Repository settings.

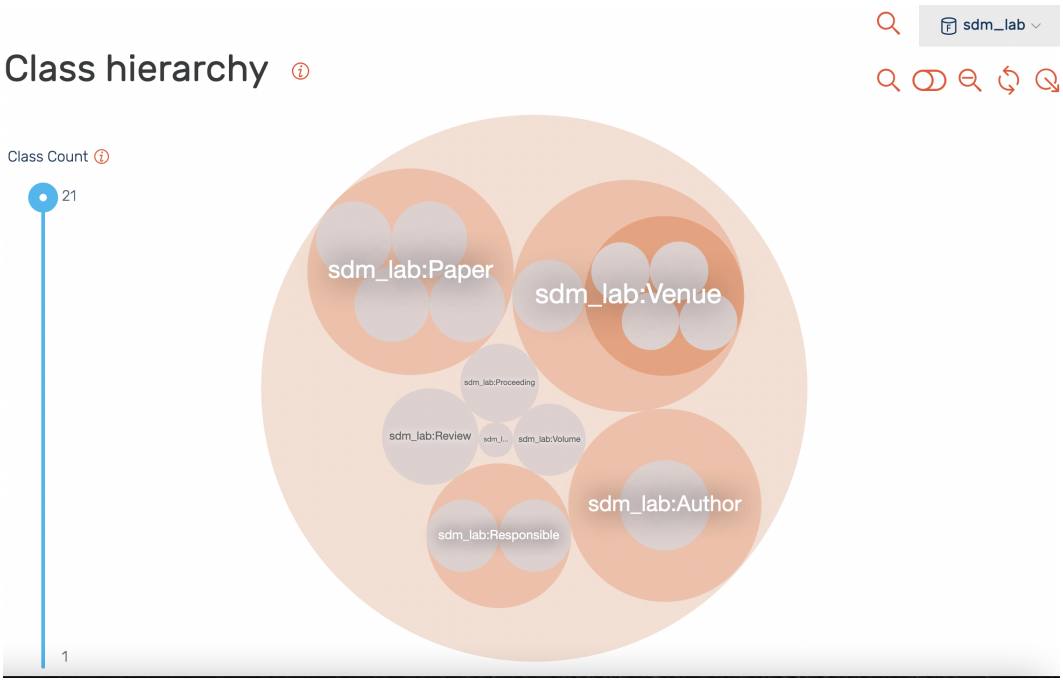
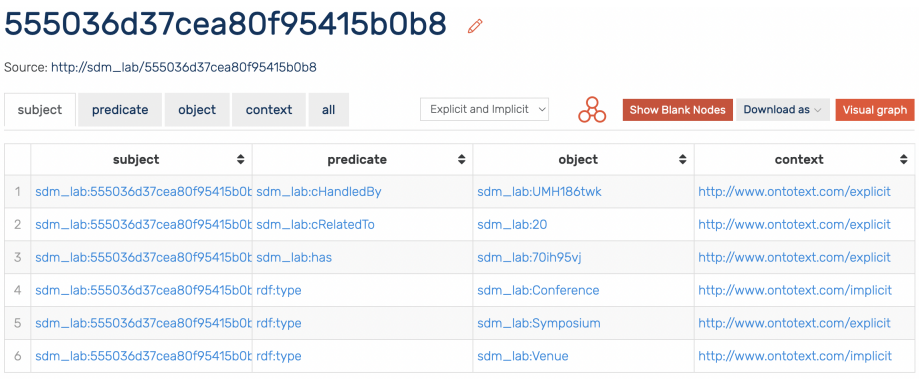


Figure 3: Class Hierarchy.

Thanks to inference, given the domain and range of our properties defined in the TBox, classes were inferred by RDFS as it can be seen by analysing for example an instance of Symposium: we can see how the Symposium class is explicit but its superclasses Conference and Venue were implicitily inferred.
This is shown in the screenshots below, the first one showing an instance of Symposium which was inferred to be a Conference and a Venue (two superclasses of Symposium); the second one showing an instance of Chair who was not only inferred to be a Responsible (superclass) but also a Chair, which was inferred from the relationship 'cHandledBy' whose range is a Chair node:



	subject	predicate	object	context
1	sdm_lab:555036d37cea80f95415b0b	sdm_lab:cHandledBy	sdm_lab:UMH186twk	http://www.ontotext.com/explicit
2	sdm_lab:555036d37cea80f95415b0b	sdm_lab:cRelatedTo	sdm_lab:20	http://www.ontotext.com/explicit
3	sdm_lab:555036d37cea80f95415b0b	sdm_lab:has	sdm_lab:70ih95vj	http://www.ontotext.com/explicit
4	sdm_lab:555036d37cea80f95415b0b	rdf:type	sdm_lab:Conference	http://www.ontotext.com/implicit
5	sdm_lab:555036d37cea80f95415b0b	rdf:type	sdm_lab:Symposium	http://www.ontotext.com/explicit
6	sdm_lab:555036d37cea80f95415b0b	rdf:type	sdm_lab:Venue	http://www.ontotext.com/implicit

Figure 4: Classes implicitily inferred.

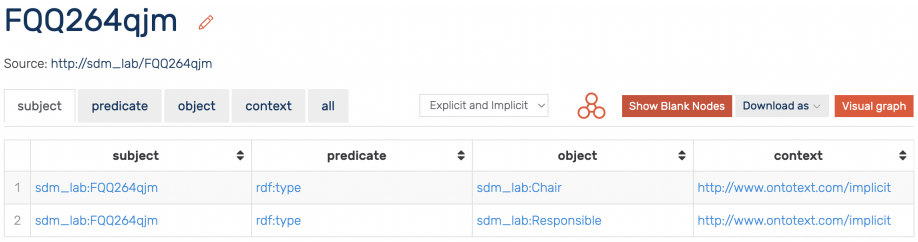


Figure 5: Chair implicitily inferred.

In addition, looking at the overall statistics of our repository we can notice the expansion ratio, given by the explicit instances over the total, is 1.33, with a total of 40,281 inferred statements. This is documented in the screenshot below.

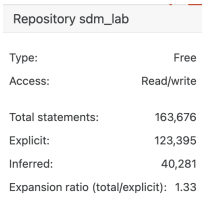


Figure 6: Expansion ratio of ontology.

3.1 Summary Statistics

After loading the TBOX and ABOX together on our repository in GraphDB, the following statistics regarding the data were obtained:

Label	Count
Number of Classes	21
Number of Main Properties	12
Number of Instances of Main Classes	41,538
Number of Triples	163,676

Statistics for individual main Clasess:

Class	Instances	Percentage
Review	13,578	32.68%
Author	10,252	24.68%
Paper	6,789	16.34%
Venue	4,796	11.54%
Responsible	2,489	5.99%
Proceeding	2,353	5.66%
Volume	1,251	3.01%
Keyword	30	0.07%

4 B.4 Querying the ontology

In addition to run the queries, .csv files of the results were downloaded and can be found in the 'query_results' folder attached to this submission.

1. Find all Authors.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX sdm_lab: <http://sdm_lab/>
SELECT ?author ?name WHERE {
   ?author rdf:type sdm_lab:Author.
    ?author sdm_lab:name ?name.
}
```

Successful run of the query on our graph via GraphDB is shown in the screenshot below:

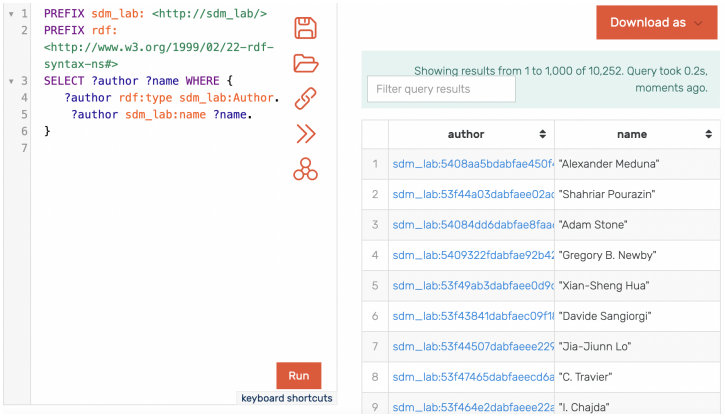


Figure 7: Query 1.

2. Find all properties whose domain is Author.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sdm_lab: <http://sdm_lab/>
SELECT ?p WHERE {
    ?p rdfs:domain sdm_lab:Author
}
```

Successful run of the query on our graph via GraphDB is shown in the screenshot below:

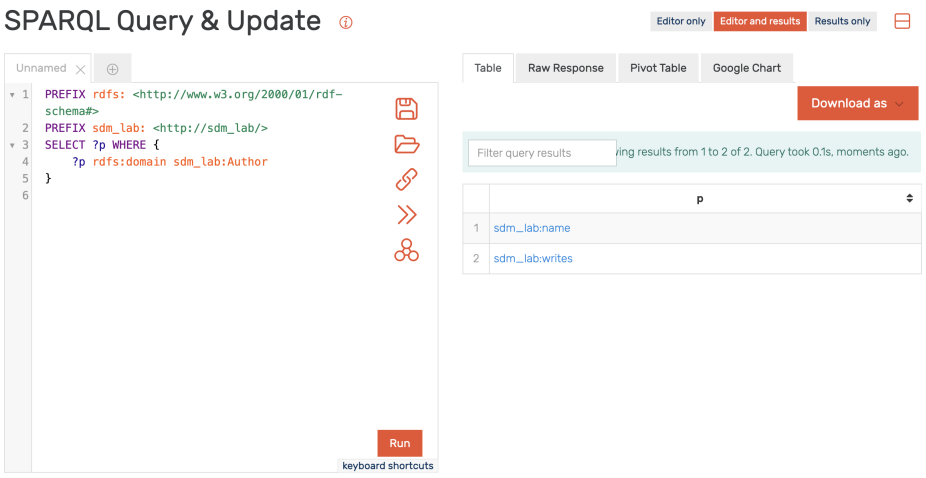


Figure 8: Query 2.

3. Find all properties whose domain is either Conference or Journal.

```
PREFIX sdm_lab: <http://sdm_lab/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?p WHERE {
    {?p rdfs:domain sdm_lab:Conference} UNION {?p rdfs:domain
    sdm_lab:Journal}
}
```

Successful run of the query on our graph via GraphDB is shown in the screenshot below:

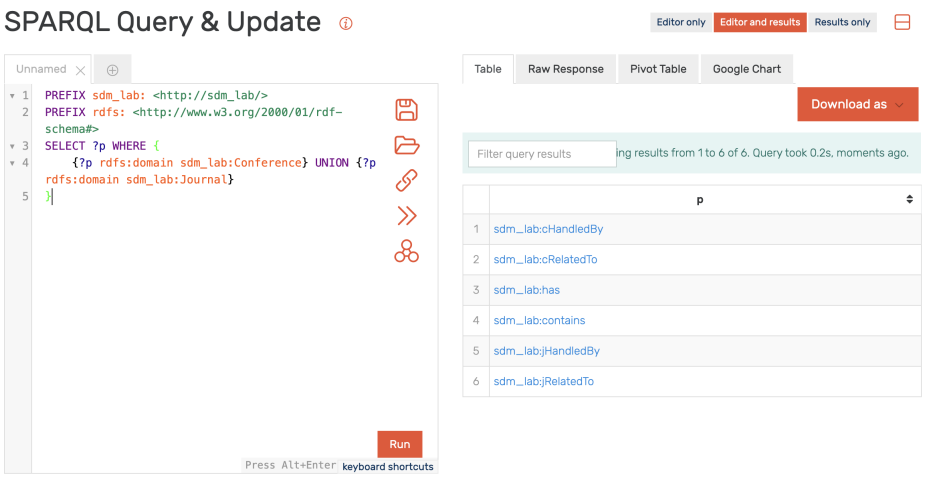


Figure 9: Query 3.

4. Find all the papers written by a given author that where published in database conferences. The author is given as an input in the query with its unique identifier (as two authors might have the same name).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sdm_lab: <http://sdm_lab/>
SELECT ?authorName (group_concat(?title;separator=",\n ") as ?titles)
WHERE {
    VALUES ?author { sdm_lab:53f4b4d4dabfaedd74eba537}.
    ?author sdm_lab:Name ?authorName.
    ?author sdm_lab:Writes ?paper.
    ?paper sdm_lab:Title ?title .
    ?paper sdm_lab:IsSubmittedTo ?conference .
    ?conference rdf:type sdm_lab:Conference .
    ?conference sdm_lab:CRelatedTo ?keyword .
    ?keyword sdm_lab:word "database".
}
GROUPBY ?authorName
```

Successful run of the query on our graph via GraphDB is shown in the screenshot below:

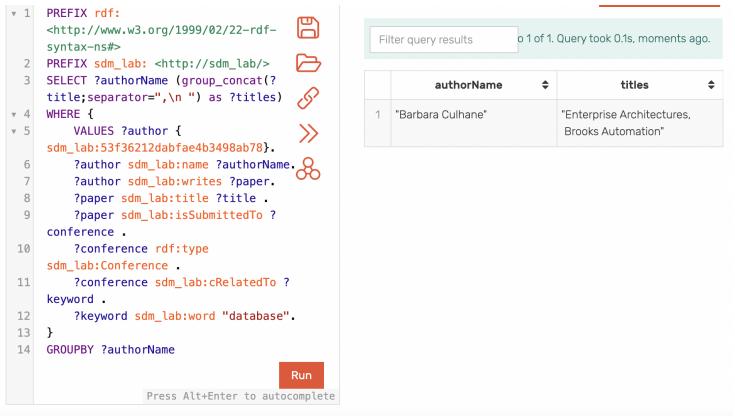


Figure 10: Query 4.

References

GraphDB. Reasoning, 2022. URL https://graphdb.ontotext.com/documentation/standard/reasoning.html. 3

RDFLib. Rdflib documentation, 2022. URL https://rdflib.readthedocs.io/en/stable/. 1