A SPARQL to Cypher Transpiler: Proposal and Initial Results

Lakshya A Agrawal[*]
lakshya18242@iiitd.ac.in
IIIT-Delhi
India

Nikunj Singhal[*]
nikunj18249@iiitd.ac.in
IIIT-Delhi
India

Raghava Mutharaju
raghava.mutharaju@iiitd.ac.in
IIIT-Delhi
India

ACM Reference Format:
Lakshya A Agrawal, Nikunj Singhal, and Raghava Mutharaju. 2022. A SPARQL to Cypher Transpiler: Proposal and Initial Results. In *5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD) (CODS-COMAD 2022), January 8–10, 2022, Bangalore, India.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3493700.3493757

1 INTRODUCTION

A Knowledge Graph [3] is defined as a graph of data intended to capture the knowledge of the real world. The nodes in the graph represent the entities of interest and the edges capture the relationship between the entities. Therefore, a Knowledge Graph structures the information (knowledge) so that it is connected and makes it easy to query information linked to one another. Knowledge Graphs are used in several applications such as chatbots, question answering systems and recommendation systems across multiple domains. The two most common representations for Knowledge Graphs are triples (RDF Graphs) and Property Graphs (PG), with each having different strengths, environments and usage scenarios. RDF and its corresponding query language, SPARQL, are W3C standards that support reasoning. Property Graphs, on the other hand, offer greater flexibility in representing the data in the form of graphs. Depending on the PG management system, there are several options to query PGs such as Gremlin, Cypher and GQL, with Cypher being one of the popular query languages of choice.

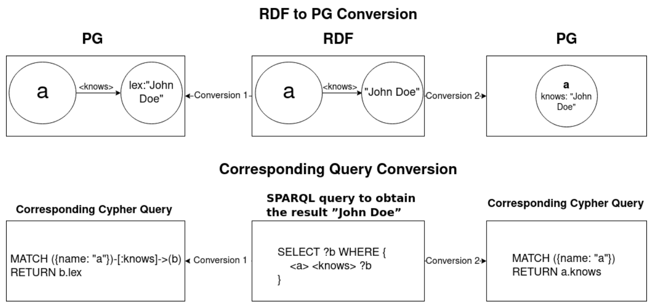
Many commercial and open-source applications built on either RDF or Property Graphs cannot leverage the data from one another due to the lack of interoperability. There is a need to interoperate between these two Worlds so that it is easy to switch with a change in the environment or the usage scenario. The ability to use SPARQL or Cypher to query Property Graphs or RDF Graphs, respectively, enables the applications to easily interoperate between the two Worlds. Although there are techniques to convert RDF Graphs to Property Graphs [2], there are no existing tools to convert SPARQL to Cypher. In order to fill this gap, we propose a transpiler (a source to source compiler) to convert SPARQL queries to Cypher queries. This will allow the data and the queries to be easily ported from an RDF graph structure to a PG. We present a three-phase approach for SPARQL to Cypher transpiler development. We also discuss an extendable testbench that supports automatic testing of the SPARQL to Cypher converter. We establish the dependence of the query transpilation on the scheme used for mapping RDF Graphs to Property Graphs. The source code is available under an open-source license at https://github.com/kracr/sparql-cypher-transpiler.

[*]Lakshya A Agrawal and Nikunj Singhal contributed equally to this work and both of them are undergraduates.

CODS-COMAD 2022, January 8–10, 2022, Bangalore, India
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8582-4/22/01.
https://doi.org/10.1145/3493700.3493757

2 APPROACH

The primary use case that we consider for the transpiler is that of enabling interoperability between RDF graph and PG by facilitating a query conversion mechanism. So we assume the presence of an RDF graph and its equivalent PG. Consider an RDF graph, G_r, and a set of SPARQL queries, Q_r, that work on G_r. In order to port them to a PG store, G_r should be mapped to its equivalent PG, G_p, and Q_r should be converted to Cypher queries, Q_p. There are multiple equivalent graph conversions from RDF to PG. The process of converting a SPARQL query to its equivalent Cypher query depends on the scheme used for converting an RDF Graph to PG as illustrated in Figure 1. Taking this into account, we propose a three-phase approach for developing a SPARQL to Cypher transpiler.



```
MATCH ({name: "a"})-[:knows]->(b)
RETURN b.lex
```

```
SELECT ?b WHERE {
    <a> <knows> ?b
}
```

```
MATCH ({name: "a"})
RETURN a.knows
```

Figure 1: RDF Graph and its conversion to PG. The Cypher translation of the SPARQL query to obtain the same result in the example - "John Doe" - is dependent on the graph conversion scheme.

2.1 Approach for Transpiler Development

The following are the three phases in the transpiler development.

(1) Build the SPARQL to Cypher converter with respect to a custom-defined RDF to PG conversion scheme so as to focus on the query conversion and not on the graph conversion.
(2) Modify the SPARQL to Cypher transpiler in phase 1 to work with existing RDF to PG conversion schemes used in PG-Bench [4], and Neosemantics [1].
(3) Build a generic SPARQL to Cypher transpiler that does not assume any particular graph conversion scheme and instead takes the graph conversion scheme as an input parameter in the query conversion process, making the transpiler independent of the graph conversion scheme.

2.2 Testbench

We developed a testbench to test the query conversion and validity with support for changing the underlying graph conversion schemes and hence independent of the RDF to PG conversion. The testbench is aimed to provide fast feedback and exhaustively test the transpiler over various datasets and language constructs. As shown in Figure 2, each testcase in the testbench consists of a RDF dataset and SPARQL query. The RDF Graph is converted to PG and SPARQL to Cypher, and the results of executing the queries on their respective datasets are then evaluated for equivalence. The RDF to PG conversion is used to ensure the correctness of the transpiler by evaluating the equivalence of results of the SPARQL and converted Cypher queries.

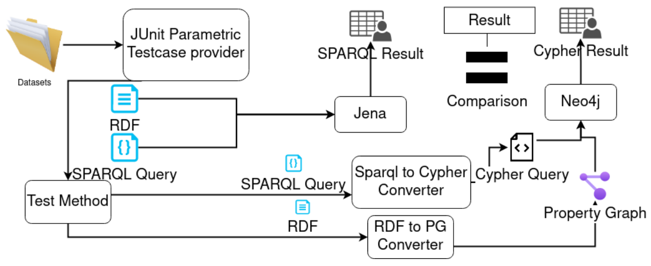


Figure 2: Testbench for our SPARQL to Cypher transpiler

2.3 RDF to PG Conversion

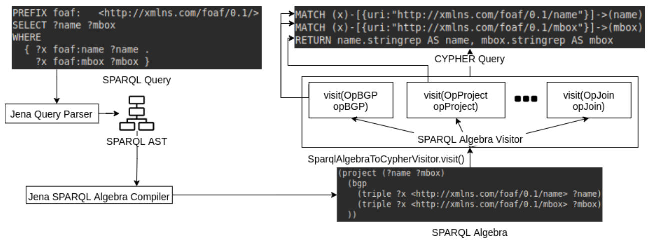
The custom RDF to PG converter for phase one iterates over each triple in an RDF dataset and creates a node for the subject and the object in the triple. The predicate in the triple becomes the edge label in the PG. A *MERGE* query corresponding to the subject and object of each triple is created using the visitor design pattern. The visitor pattern is used to exhaustively cover all the RDF node types - blank node, IRIs and literals. *MERGE* query in Cypher matches the node pattern in the PG, and upon non-existence, creates it.

2.4 SPARQL to Cypher Transpiler

The first step in the conversion is to get the corresponding algebra for a given SPARQL query. We make use of Apache Jena[1] to obtain the SPARQL algebra. The Cypher query is built incrementally by using the visitor design pattern to exhaustively visit all the SPARQL algebra nodes (Figure 3). Currently, the following types of node visits are supported.

(1) Projection (Jena.OpProject): In SPARQL algebra, projections represent variable selection. In Cypher, the corresponding selections are performed through a return clause, which returns a string representation for the IRI associated with each node/edge.
(2) Basic Graph Pattern (Jena.OpBGP): BGPs are converted using a node visitor along with a visitor for named and blank variables in SPARQL queries. Each variable in a SPARQL query must be converted to a variable in Cypher, which should refer to the PG entity corresponding to the RDF entity. Blank variables and named variables are mapped to legal Cypher names, with a check for name collision and the mapping is retained throughout the conversion.

[1]https://jena.apache.org/documentation/query/algebra.html


```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
```

```
MATCH (x)-[{uri:"http://xmlns.com/foaf/0.1/name"}]->(name)
MATCH (x)-[{uri:"http://xmlns.com/foaf/0.1/mbox"}]->(mbox)
RETURN name.stringrep AS name, mbox.stringrep AS mbox
```

```
(project (?name ?mbox)
  (bgp
    (triple ?x <http://xmlns.com/foaf/0.1/name> ?name)
    (triple ?x <http://xmlns.com/foaf/0.1/mbox> ?mbox)
  ))
```

Figure 3: Converting a SPARQL query to its equivalent Cypher query

3 EVALUATION

The transpiler is evaluated on the testbench. The RDF graph and the SPARQL query sources are as follows.

- SP^{2}Bench [5]: We generate 100 triples and use the 17 SPARQL queries that are part of the benchmark.
- W3C SPARQL Recommendation Document[2]: We make use of 5 RDF Graphs and 10 queries from this document.

The converted queries for all supported query types return the same results as the corresponding SPARQL query and hence pass the test.

4 CONCLUSION AND FUTURE WORK

Going forward, we plan to work on the remaining phases of our proposed three-phase approach. For the third phase, a mechanism to capture the RDF Graph to PG conversion scheme needs to be investigated. We plan to develop a simple mapping language to capture the conversion. The testbench will be extended to support query performance benchmarking, which will further be used to work on query optimizations. We also plan to work on formal algebraic arguments to prove the correctness of the query conversion process. This would require further study of the algebraic formalization of Cypher.

Acknowledgement. This work is partially supported by the Infosys Centre for Artificial Intelligence (CAI), IIIT-Delhi, India.

REFERENCES

[1] Jesús Barrasa and Adam Cowley. 2021. neosemantics (n10s): Neo4j RDF Semantics toolkit - Neo4j Labs. https://neo4j.com/labs/neosemantics/
[2] Olaf Hartig. 2014. Reconciliation of RDF* and Property Graphs. *CoRR* abs/1409.3288 (2014), 1–18. arXiv:1409.3288 http://arxiv.org/abs/1409.3288
[3] Aidan Hogan, Eva Blomqvist, Michael Cochez, and et al. 2021. Knowledge Graphs. *Comput. Surveys* 54, 4 (Jul 2021), 1–37. https://doi.org/10.1145/3447772
[4] Meenakshi Maindola and Raghava Mutharaju. 2020. *PGBench: A Property Graph Benchmark for Knowledge Graphs*. Master's thesis. Indraprastha Institute of Information Technology, Delhi.
[5] Michael Schmidt, Thomas Schallhorn, Georg Lausen, and Christoph Pinkel. 2009. SP2Bench: A SPARQL Performance Benchmark. In *2009 IEEE 25th International Conference on Data Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 222 – 233. https://doi.org/10.1109/ICDE.2009.28

[2]https://www.w3.org/TR/sparql11-query/