KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

DE NAYER (SINT-KATELIJNE-WAVER) CAMPUS

Inverting knowledge graphs back to raw data

How can we leverage the rules we use to construct knowledge graphs to do the inverse?

Tijs VAN KAMPEN

Promotor: Prof dr. ir. Anastasia Dimou

Master thesis submitted to obtain the degree of Master of Science in the engineering technology: Electronics-ICT

academic year 2023 - 2024

Contents

Chapter 1

Introduction

The earliest academic definition of a knowledge graph can be found in a 1974 article as

> A mathematical structure with vertices as knowledge units connected by edges that represent the prerequisite relation (Marchi and Miguel, 1974; Bergman, 2019)

The idea of expressing knowledge in a graph structure predates even this definition, with the concept of semantic networks (Richens, 1956). However, the term knowledge graph only became well-known after Google announced they were using a knowledge graph to enhance their search engine in 2012 (Singhal, 2012). Knowledge graphs are used to make search engines, chatbots, question answering systems, etc more intelligent by injecting knowledge into them (Ji et al., 2022).

A knowledge graph consists of many connected nodes, where each node is either an entity or a literal. These nodes are connected by edges, where each edge defines a relation between two nodes. RDF is a framework often used to represent knowledge graphs, it uses subject-predicate-object triples to represent the nodes and their edges. Every node is either an URI, a blank node or a literal. The edges are URIs. A triple representing the fact that the entity `John Doe` has the first name `John` would look like this: `http://example.com/John_Doe http://schema.org/givenName "John"`. Often the predicates are chosen from an ontology/vocabulary, such as schema.org or FOAF. This allows for more interoperability between knowledge graphs, as the same predicates are used to represent the same concepts.

These knowledge graphs are constructed by extracting information from various sources, both unstructured sources such as text (using natural language processing) and (semi-)structured sources such as databases, CSV, XML, JSON, RDF (using mapping languages). Many mapping languages exist, differing on the way of defining the rules and the target source file format. Some mapping languages use the turtle syntax, while others provide their own custom syntax, and others repurpose existing languages like SPARQL or ShEx. (Van Assche et al., 2023). Some languages are specific to a single source format, such as R2RML(turtle format) (Das et al., 2012) for relational databases, XSPARQL(SPARQL format) (Bischof et al., 2012) for XML. Others can process multiple formats, such as RML (turtle) (Dimou et al., 2014), D-REPR (YAML), xR2RML (turtle), etc. These have the ability to map from multiple sources in different formats.

To achieve this these mapping languages use a declarative approach, where the user specifies the mapping rules, and the implementation of the mapping language takes care of the actual mapping. Two ways of doing the mapping exist: materialisation and virtualisation. Materialisation constructs the knowledge graph as a file, which can be loaded into a triple store. Virtualisation does not generate the knowledge graph as a file, but instead exposes a virtual knowledge graph, which can be queried as if it were a real knowledge graph. (Calvanese et al., 2017).

Creating these mapping rules is often done by hand. There are tools that make creating these mappings easier, like RMLEditor (Heyvaert et al., 2018b) which exposes a visual editor and YARRRML (Heyvaert et al., 2018a) which allows users to create rules in the user-friendly YAML which are then compiled to RML rules. Alternatively tools are starting to be created for automatic generation of mapping rules from e.g. SHACL.

Retrieving data from a knowledge graph, for consumption by other programs, is done by querying the knowledge graph using SPARQL (Seaborne and Prud'hommeaux, 2008) for tabular data and XSPARQL (Bischof et al., 2012) or XSLT for XML. XSPARQL is the only language that can both map[/lift] and query[/lower], but the syntax for mapping and querying differs, so it could be argued that XSPARQL is actually two languages.

We can not convert the knowledge graph back to the original data format using the same rules we created it with. As such any changes we make to the data are hard to propagate back to the original data. We can not update, expand or improve the original data using e.g. knowledge graph refining. Nor can we apply changes to a virtual knowledge graph to change the original data.

In thesis we seek to answer the question: *How can we extend an existing system like RML or create a new system to construct raw data from knowledge graphs?* We choose to extend the Morph-KGC implementation (Arenas-Guerrero et al., 2022) of RML (Dimou et al., 2014) as RML's end-to-end (from file to knowledge graph) characteristics make it a good candidate for this task. To answer the main research question we need to answer the following sub-questions:

RQ1 *How can we construct the schema of the original data from the mapping rules?*
RQ2 *How can we populate the schema with data from the knowledge graph?*

1.1 Thesis outline

This thesis aims to explore the possibility of inverting knowledge graphs back to their original data format using RML mapping rules. To achieve this we will first look at the current state of the art in chapter 2. We will take a closer look at the technologies used like RDF, SPARQL, and RML. We will also look at the current state of the art for inverting knowledge graphs. In chapter 3 we will look at our implementation of the inversion algorithm. We will look at the algorithm itself, and the implementation details. In chapter 4 we will evaluate our implementation using various benchmarks. For basic testing we use a subset of the rml test cases, which are designed to test the conformance of tools to the RML specification. For more advanced testing we will use various benchmarks simulating real-life use cases like LUBM4OBDA, GTFS-Madrid-Bench and SDM-Genomic-dataset. Finally in chapter 5 we will conclude this thesis, and look at possible future work.

Chapter 2

Related work

Introduction text providing an overview of the related work

2.1 Semantic Web

Tim Berneers-Lee envisioned a version of the web that would also be understandable by machines, and thus the semantic web was born. It is not designed as a separate entity to the web, but instead as an extension, mostly hidden for normal humans. It is designed with mostly existing technologies like XML(including HTML, being a superset of it), URI and RDF. Even ontologies, a key component of the semantic web, are not a new concept, but rather co-opted from the field of philosophy. (Berners-Lee et al., 2001)

2.2 RDF

RDF was originally designed as a data model for metadata, but has been extended to be a general purpose framework for graph data. RDF is a directed graph, where the nodes represent entities, and the edges represent relations between these entities. This graph is build up from triples, which connect a subject and an object using a predicate as shown in figure 2.1.

The subject must always be an entity, which can either be represented by an URI or be a blank node. The predicate must be an URI, and the object can be either an URI, a blank node or a literal. (Manola and Miller, 2004)

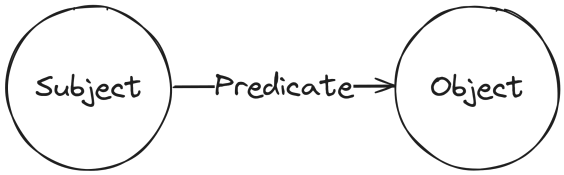

Figure 2.1: An RDF triple

Subject	Predicate	Object
http://example.com/De_Nayer	https://schema.org/location	http://example.com/Sint_Katelijne_Waver
http://example.com/r0785695	https://schema.org/givenName	“Tijs”

Figure 2.2: Example of RDF triples

Subject	Predicate	Object
ex:student/r0785695	schema:address	_:addrr0785695
_:addrr0785695	schema:postalCode	”2800”^^xsd:integer
_:addrr0785695	schema:streetAddress	”Gentsesteenweg 2705”@nl

Figure 2.3: Example of a blank node and prefix notation

An URI is a unique identifier for a resource on the web. Unlike a normal URL, it does not have to point to a network location, but can also be used to identify a person, a location, a concept, etc. (Manola and Miller, 2004). In RDF the URI is purely used for identifying resources. As such, unlike in HTML where certain conventions are expected, there are no conventions for URIs in RDF. An example of this can be seen in figure 2.2, the example subjects share the same domain but this does not imply that they are closely related, or even related at all. URIs are extended to IRIs to allow for a wider range of characters. Except for allowing unicode characters, IRIs are identical to URIs so little distinction is made between the two in this thesis.

A blank node is a node that is not identified by a URI. It is used to represent an anonymous resource that can’t be or has no reason to be uniquely identified. For example the address of student r0785695 in figure 2.3 is only pertinent to the student, and thus does not need to be uniquely identified. A blank node is serialized as `_:name`, where name is a unique identifier for the blank node. This identifier is only unique within the document, and thus can’t be used to refer to the blank node from outside the document. (Manola and Miller, 2004)

A literal is a value, e.g. a string, integer or date. This value can be typed, e.g. a string can be typed as a date, or untyped. A string can also have a language tag, which is used to indicate the language of the literal.

RDF is only a framework, and as such does not define any serialization syntax. There are however a few common serialization standards like for example RDF/XML, Turtle, N-Triples and JSON-LD.

2.2.1 Turtle

Turtle, or Terse RDF Triple Language, is a human-readable serialization format for RDF. It is the most used serialization format for RDF, and is used in many tools and specifications. In its simplest form turtle consists of triple statements, sequences of subject-predicate-object separated by spaces and terminated by a dot. An example of this can be seen in listing 2.1. This is very verbose, but turtle offers many features to make it more concise. Below is a list of some of these features and, if possible, how they can be used to make the example more concise.

- **Prefix notation** allows us to shorten URIs by defining a prefix.
  - Using `@prefix schema: <https://schema.org/>` allows us to shorten `https://schema.org/Person` to `schema:Person`
- **Base prefix** allows us to shorten URIs by defining a base URI.
  - Using `@base <http://example.com/>` allows us to shorten `http://example.com/r0785695` to `r0785695`
- **Predicate lists** allow us to shorten multiple triples with the same subject to a list of predicates.
  - All our triples in the example are about r0785695, so we can split the predicates with `;` instead of repeating the subject.
- **Object lists** allow us to shorten multiple triples with the same subject and predicate to a list of objects.
  - r0785695 is both a Person and a Student, so we can split the objects with `,` instead of repeating the subject and predicate.
- **Literals** allow indentify values, e.g. strings, integers, dates, etc. with a datatype or language tag.
- **Blank nodes** allow us to define anonymous resources by using the `_:` prefix.
- **Unlabeled blank nodes** allow us to define anonymous resources without a unique identifier by using the [] notation instead of `_:name`.
- **Collections** allow us to define a list of blank nodes by using the () notation.

The example in listing 2.1 can be rewritten using these features, as shown in listing 2.2.

```
<http://example.com/r0785695> <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#type> <http://schema.org/Person> .
<http://example.com/r0785695> <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#type> <http://schema.org/Student> .
<http://example.com/r0785695> <http://schema.org/givenName> "Tijs" .
<http://example.com/r0785695> <http://schema.org/familyName> "Van
    Kampen" .
```

Listing 2.1: Basic naive turtle document

```
@prefix schema: <https://schema.org/> .
@base <http://example.com/> .
<r0785695> a schema:Person, schema:Student ;
    schema:givenName "Tijs" ;
    schema:familyName "Van Kampen" .
```

Listing 2.2: Basic turtle document using turtle features

2.3 SPARQL

SPARQL Protocol And RDF Query Language (SPARQL) is the W3C standard query language for RDF. It is the main way to query RDF data, and is similar to SQL. SPARQL queries mostly consist of a pattern of triples, which are matched against the RDF graph, a basic example can be found in listing 2.3. Querying is very feature rich, with support for aggregation, subqueries, negation, etc. It also supports different return types, federated queries, entailment, etc. (Harris and Seaborne, 2013) Aside from the query protocol it also defines the graph store protocol, which can be used to manipulate graph databases directly (Aranda et al., 2013). *We could go into more detail here on each or any of these features, but getting any decent overview would take pages and pages. We dont really use any of these fancy features in our work anyways, if that changes we will add more detail here.*

```
PREFIX schema: <https://schema.org/>
SELECT ?name ?address
WHERE {
    ?student schema:givenName ?name .
    ?student schema:address ?address .
}
```

Listing 2.3: Example of a basic SPARQL SELECT query

2.4 Mapping languages

Mapping languages are used to define a mapping between a source and a target. The target in the context of linked data is of course RDF, with the source being any structured data source. Some mapping languages exist for a single source, e.g. Relational Database to RDF Mapping Language (R2RML) for relational databases, a query language combining XQuery and SPARQL (XSPARQL) for XML, etc. Others are more general purpose, e.g. RDF Mapping Language (RML) and Data to RDF Mapping Language (D2RML). We will discuss both R2RML and RML in more detail, as one extends the other. RML we will discuss because it is one of the more feature rich general mapping languages, and it is the mapping language we will use in our implementation. R2RML we discuss because it is the most widely used mapping language, as it supports virtualization ontop of databases. *i wonder if people will get the pun in the previous sentence* Most RML implementations also support R2RML, as RML is a nearly superset of R2RML.

2.4.1 R2RML

Relational Database to RDF Mapping Language (R2RML) is a mapping language for mapping relational databases to RDF. As opposed to Direct Mapping (DM), which results in a direct mapping from the relational database to RDF without any changes to structure or naming, R2RML allows for more flexibility. R2RML mappings consist of zero or more TriplesMaps, which are used to map a

table to RDF. A TriplesMap consists of a logical table, a subject map, and one or more predicate object maps (POMs).

The logical table is used to define the table that is being mapped, with each row in the table being mapped to a subject and it's corresponding POMs. It is possible to create a view of a table by using a SQL query, and then map this view. This allows for more complex mappings, e.g. mapping a join of two tables or a computed column.

Each of the SubjectMap, PredicateMap, ObjectMap, (and GraphMap) is a subclass of TermMap, which is a function that generates an RDF term. The map type can be constant, template, or column. The resulting term is then used as the subject, predicate, object, or graph of the triple. The termType of the map determines the type of the term, which can be IRI, blank node or literal. If the termType is literal, optionally the datatype or language can be added. In accordance with the RDF specification, not all combinations of termType and map are possible, this is shown in table 2.1. The object map has an additional subclass, a reference object map, in which we refer to another TriplesMap. Using a reference map we can map a foreign key to the subject of another TriplesMap, with a join condition. (Cyganiak et al., 2012)

TermType	Subject	Predicate	Object	Graph
IRI	✓	✓	✓	✓
Blank node	✓	✗	✓	✓
Literal	✗	✗	✓	✗

Table 2.1 Possible combinations of TermType and Map type

A constant value is a fixed value, e.g. a URI or a string. A template is a string with placeholders, which are replaced by values from the logical row. A column is the value of a column in the logical row.

```
    @prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "EMP" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{EMPNO}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:id;
        rr:objectMap [ rr:column "EMPNO"; rr:datatype xsd:positiveInteger ].
    ].
```

Listing 2.4: Example of an R2RML mapping

R2RML		RML	
Logical Table (relational database)	rr:logicalTable	Logical Source	rml:logicalSource
Table Name	rr:tablename	URI (pointing to the source)	rml:source
column	rr:column	reference	rml:reference
(SQL)	rr:SQLQuery	Reference Formulation	rml:referenceFormulation
per row iteration		defined iterator	rml:iterator

Table 2.2 Differences between R2RML and RML

2.4.2 RML

RDF Mapping Language (RML) is a mapping language for mapping any (semi-)structured data source to RDF. It is a generalization of R2RML, and as such supports all the features of R2RML. It extends R2RML by extending database specific features to make them more general. The differences in usage can be seen in table 2.2. (Meester et al., 2022)

RML uses the same structure as R2RML, with TriplesMaps consisting of a logical source, a subject map, and zero or more POMs. The changes it has all relate to the logical source. Whereas in R2RML the source is always a database, from which we select a table or view, in RML the source can be one of many different source types like XML, JSON, CSV, etc. Where in R2RML we simply iterate over the rows of a table, in RML we can have a source without an explicit iteration pattern and as such we need to define an iterator.

Chapter 3

Implementation

Chapter 4

Evaluation

In this chapter we will evaluate our implementation. We will do this by testing it again various datasets, comparing the expected results with the actual results. For each dataset we will go over our testing methodology and its results.

4.1 RML test cases

The RML test cases are a set of test cases to evaluate the conformity of an RML processor. Though these test cases are not a perfect match, they offer expected outputs for certain inputs and mapping rules, making them a good starting point for testing our implementation. Tests that error do not produce a result, so we start by filtering those out.

Chapter 5

Conclusion

Placeholder chapter for refering to the conclusion chapter. This should be removed in the intermediate report.

Bibliography

Aranda, C. B., Corby, O., and Das, S. (2013). SPARQL 1.1 overview. W3C recommendation, W3C. https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., Pérez, M. S., and Corcho, O. (2022). Morph-KGC: Scalable knowledge graph materialization with mapping partitions. *Semantic Web*.

Bergman, M. K. (2019). A common sense view of knowledge graphs. *AI3:::Adaptive Information*, 1(1):1–1.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5).

Bischof, S., Decker, S., Krennwallner, T., Lopes, N., and Polleres, A. (2012). Mapping between rdf and xml with xsparql. *Journal on Data Semantics*, 1(3):147–185.

Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., and Xiao, G. (2017). Ontop: Answering sparql queries over relational databases. *Semantic Web*, 8:471–487. 3.

Chortaras, A. and Stamou, G. (2018). D2rml: Integrating heterogeneous data and web services into custom rdf graphs. In *LDOW@WWW*.

Cyganiak, R., Sundara, S., and Das, S. (2012). R2RML: RDB to RDF mapping language. W3C recommendation, W3C. https://www.w3.org/TR/2012/REC-r2rml-20120927/.

Das, S., Cyganiak, R., and Sundara, S. (2012). R2RML: RDB to RDF mapping language. W3C recommendation, W3C. https://www.w3.org/TR/2012/REC-r2rml-20120927/.

Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., and Van de Walle, R. (2014). RML: a generic language for integrated RDF mappings of heterogeneous data. In Bizer, C., Heath, T., Auer, S., and Berners-Lee, T., editors, *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proceedings*.

Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. W3C recommendation, W3C. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

Heyvaert, P., De Meester, B., Dimou, A., and Verborgh, R. (2018a). Declarative rules for linked data generation at your fingertips! In Gangemi, A., Gentile, A. L., Nuzzolese, A. G., Rudolph,

S., Maleshkova, M., Paulheim, H., Pan, J. Z., and Alam, M., editors, *The Semantic Web: ESWC 2018 Satellite Events*, pages 213–217, Cham. Springer International Publishing.

Heyvaert, P., Dimou, A., De Meester, B., Seymoens, T., Herregodts, A.-L., Verborgh, R., Schuurman, D., and Mannens, E. (2018b). Specification and implementation of mapping rule visualization and editing: MapVOWL and the RMLEditor. *Journal of Web Semantics*, 49:31–50.

Ji, S., Pan, S., Cambria, E., Marttinen, P., and Yu, P. S. (2022). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514.

Manola, F. and Miller, E. (2004). RDF primer. W3C recommendation, W3C. https://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

Marchi, E. and Miguel, O. (1974). On the structure of the teaching-learning interactive process. *International Journal of Game Theory*, 3(2):83–99.

Meester, B. D., Heyvaert, P., and Delva, T. (2022). RML: RDF mapping language. Unofficial draft, RML. https://rml.io/specs/rml/.

Richens, R. H. (1956). Preprogramming for mechanical translation. *Mech. Transl. Comput. Linguistics*, 3:20–25.

Seaborne, A. and Prud'hommeaux, E. (2008). SPARQL query language for RDF. W3C recommendation, W3C. https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

Singhal, A. (2012). Introducing the knowledge graph: things, not strings. 2020-11-13.

Van Assche, D., Delva, T., Haesendonck, G., Heyvaert, P., De Meester, B., and Dimou, A. (2023). Declarative rdf graph generation from heterogeneous (semi-)structured data: A systematic literature review. *Journal of Web Semantics*, 75:100753.