Towards Algebraic Mapping Operators for Knowledge Graph Construction

Sitt Min Oo[1,*], Ben De Meester[1], Ruben Taelman[1] and Pieter Colpaert[1]

[1]IDLab, Department of Electronics and Information Systems, Ghent University – imec, Technologiepark-Zwijnaarde 122, 9052 Ghent, Belgium

Abstract

Declarative knowledge graph construction has matured to the point where state of the art techniques are focusing on optimizing the mapping processes. However, these optimization techniques use the syntax of the mapping language without considering the impact of the semantics. As a result, it is difficult to compare different engines fairly due to the obscurity in their semantic differences. In this poster paper, we propose an initial set of algebraic mapping operators to define the operational semantics of mapping processes, and provide a first step towards a theoretical foundation for mapping languages. We translated a series of RML documents to algebraic mapping operators to show the feasibility of our approach. We believe that further pursuing these initial results will lead to greater interoperability of mapping engines and languages, intensify requirements analysis for the upcoming RML standardization work, and an improved developer experience for all current and future mapping engines.

1. Introduction

Several mapping engines exist to generate RDF Knowledge Graphs (KG) from heterogeneous data sources [1, 2, 3]. Each mapping engine has its own operational semantics depending on the software architecture and the mapping language it supports. This leads to redundant implementation of similar operations and incompatibility with the other engines, especially in terms of optimization techniques. For example, SDM-RDFizer [1] relies on *Triples Maps* (an RML concept [4]) to optimize deduplication and joins, which is incompatible with SPARQL-Generate [3] where SPARQL is being used (i.e. no notion of Triples Maps).

In the domain of knowledge graph querying, *algebraic operators* form the foundation for the query semantics through formalization [5]. Semantic formalization enables a) execution consistency across different query engine implementations, b) identification of redundant and contradicting notions, c) analysis of complexity and expressiveness, and d) more portable algorithms, enabling easy inheritance from existing algorithms with similar semantics. Thus, having an equivalent set of algebraic operators for the mapping process will lay the foundation to formalize the mapping process, clarifying the operational semantics and improving the

ISWC 2023 Posters and Demos: 22nd International Semantic Web Conference, November 6–10, 2023, Athens, Greece
*Corresponding author.
x.sittminoo@ugent.be (S. M. Oo); ben.demeester@ugent.be (B. D. Meester); ruben.taelman@ugent.be (R. Taelman); pieter.colpaert@ugent.be (P. Colpaert)
0000-0002-0877-7063 (S. M. Oo); 0000-0001-7116-9338 (B. D. Meester); 0000-0002-9421-8566 (R. Taelman); 0000-0002-9421-8566 (P. Colpaert)

interoperability of mapping engines.

There are solutions which provide an ontology for representing the different mapping languages [6] or provide a language-independent template for RDF knowledge graph construction [7] to increase interoperability between mapping engines. Nonetheless, the aforementioned solutions do not provide a theoretical foundation for generic mapping languages since they capture the language syntax instead of the semantics.

In this poster paper, we introduce an initial set of *algebraic mapping operators* and define their semantics. We apply this initial set to RML. We translated a series of RML documents to algebraic mapping operators to validate our approach.

2. Definition

We first introduce terminologies. For our initial work, we reuse some of the terminologies from SPARQL algebra, allowing us to align our mapping algebra with SPARQL algebra in the future to study expressiveness. We can reuse following definitions. A solution mapping μ is a partial function mapping from V, a set of *variables*, to T, a set of *terms*, provided $V \cap \mathrm{T} = \emptyset$. $\mathrm{T} = I \cup B \cup L$ where I, B, and L are disjoint, infinite sets of IRIs, blank nodes, and literals respectively.

Mapping languages enable users to fragment the generated data into different data sinks (e.g. multiple files or web sockets). In order to future proof the mapping algebra, we need to introduce the *fragment*. A fragment, $f \in F$, is a *grouping* of the multiset of solution mappings. It can be seen as a generic sink: a *file*, a *database*, or a logical fragment such as a *specific social context*, e.g. information about a person only known by friends.

A *mapping tuple* ω is the core of our mapping algebra; a partial function which maps *fragments* to *multiset of solution mappings*: $\omega : F \rightarrow \Omega$ with Ω a multiset of solution mappings. A multiset of mapping tuples is ξ. This mapping of fragments to multiset of solution mappings enable us to *group* solution mappings based on some abstract concept. For example, we could have mapping tuples where solution mappings are *grouped* according to some social context (e.g. personal information and friend's information) (Table 1). Currently, grouping solution mappings according to fragments can not be achieved with SPARQL's definition of *group* algebra.

Table 1
Two mapping tuples describing information related to John. The tuples are fragmented according to *personal* information about John and information about John's friends.

	Multiset of Solution Mappings			
Fragment	Solution Mapping	?name	?age	?email
$f_{personal}$	μ_1	John Doe	23	john.doe@example.com
$f_{friends}$	μ_2	Susan Sue	25	susan.sue@example.com
	μ_3	Alice Joe	26	alice.joe@example.com

We now define the initial set of algebraic mapping operators computing on ξ: Source, Project, Extend, and Serialize. We do not yet include the *fragmentation* operator where ξ could be recursively fragmented nor the *join* operator. The algebraic mapping operators take ξ as input

and produces ξ as output unless otherwise stated.

Source operator The Source operator is needed to generate the mapping tuples from heterogeneous data sources used in the downstream operators for mapping. The source operator is the leaf node operator in the mapping plan and does not have ξ as input.

Given a configuration, C, a source operator generates a multi-set, ξ, of *mapping tuples, ω's,* where a default fragment f_0 is mapped to a multiset of solution mappings Ω. $\mu \in \Omega$ is generated by flattening the data records which are derived by iterating over the data source. For example, μ derived from a CSV row is a partial function from the headers of the CSV to the corresponding data values in the CSV row. We define:

$$\begin{aligned} f_0 &= \text{a default fragment} \\ \Omega &= \{\mu \mid \mu = \text{ flattened data record}\} \\ \text{Source(C)} &= \{\omega \mid \omega : f_0 \rightarrow \Omega\} \end{aligned} \tag{1}$$

Project operator The Project operator restricts the variables in the solution mapping, needed to efficiently process the mapping tuples. For example, RML's single iteration of CSV contains all columns for a data record, but implicitly projects the required references for the mapping process. It is similar to the SPARQL algebra counterpart. Let $\{a_0, \ldots, a_n\} \in P$, be a set of projection attributes. We define:

$$\begin{aligned} \text{Project}(\mu, \text{P}) &= \mu \text{ restricted to attribute variables in P} \\ \text{Project}(\Omega, \text{P}) &= \{\text{Project}(\mu, \text{P}) \mid \mu \in \Omega\} \\ \text{Project}(\omega, \text{P}) &= f \rightarrow \text{Project}(\Omega, \text{P}) \\ \text{Project}(\xi, \text{P}) &= \{\text{Project}(\omega, \text{P}) \mid \omega \in \xi\} \end{aligned} \tag{2}$$

Extend operator The Extend operator derives new attributes for a solution mapping. For example, to include body mass index (BMI) of a person in the output, we derive the BMI from the height and weight attributes of a person. In RML, this is equivalent to the template and constant Term Maps, where new values, not existing in the data records, are generated. The Extend operator derives a value, by executing an expression $expr$ on the solution mapping, and coupled it to new variable v not in the domain of the solution mapping. If evaluating the expression causes an error and the variable is not in the domain of the solution mapping, the extend operator behaves like an identity operator. It is undefined if the variable restriction is violated. We define:

$$\begin{aligned} \text{Extend}(\mu, \text{v}, \text{expr}) &= \mu \cup \{(v, \mathit{value}) \mid v \notin \text{dom}(\mu) \text{ and value} = \text{expr}(\mu)\} \\ \text{Extend}(\Omega, \text{v}, \text{expr}) &= \{\text{Extend}(\mu, \text{v}, \text{expr}) \mid \mu \in \Omega\} \\ \text{Extend}(\omega, \text{v}, \text{expr}) &= f \rightarrow \text{Extend}(\Omega, \text{v}, \text{expr}) \\ \text{Extend}(\xi, \text{v}, \text{expr}) &= \{\text{Extend}(\omega, \text{v}, \text{expr}) \mid \omega \in \xi\} \end{aligned} \tag{3}$$

Serialize operator The Serialize operator serializes the mapping tuples into the specified format. This is the core functionality of mapping engines: to generate data in a specific format

from some data. For example, RML defines the data format implicitly using the Term Maps. The Serialize operator is the root node operator.

The Serialize operator generates data in a specific data format by replacing query variables in the template with the values from the input solution mappings. Each solution mapping generates one data item in the specified format. Given a string template τ. We define:

$$\begin{aligned} &\text{Serialize}(\Omega,\tau) = \{\tau(\mu) \mid \mu \in \Omega, \tau(\mu) = \text{variables in } \tau \text{ substituted with } \mu\} \\ &\text{Serialize}(\omega, \tau) = f \rightarrow \text{Serialize}(\Omega, \tau) \\ &\text{Serialize}(\xi, \tau) = \{\text{Serialize}(\omega, \tau) \mid \omega \in \xi\} \end{aligned} \tag{4}$$

3. Preliminary Results

We implemented these proposed semantics in a proof-of-concept algebra interpreter for RML mapping rules: https://github.com/s-minoo/meamer-rs . We translated several RML documents (without joins) to a tree of mapping algebraic operators: a mapping plan. This provides an initial validation of our proposed semantics, and showcases its potential: multiple mapping plans of different complexity are translated, allowing for inspection and optimization proposals.

4. Conclusion

This poster paper presents an initial set of algebraic mapping operators (Source, Project, Extend, Serialize) and proof-of-concept implementation which can already be used to describe a subset of mapping processes represented in RML. The generated mapping plans show the potential to define optimization rules based on the semantics and not syntax of the mapping language. Furthermore, working with algebraic mapping operators enables us to "rewrite" the mapping plan, generated using the algebraic mapping operators, to optimize the mapping process. For example, we could push the Projection operator close to the Source operator, to filter out unnecessary data unused in the output, to reduce memory usage during the knowledge graph construction.

As future work, we plan to extend and refine the algebraic operators, to be used as a theoretical foundation for mapping languages. We plan to provide a generic mapping framework: the reference implementation of these algebraic operators. Users could use this framework to easily create a mapping engine using their choice of mapping language. Finally, we plan to conduct an empirical study on the existing mapping optimization techniques and translate them to optimization rules using these algebraic operators.

Acknowledgments

The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10), and the imec ICON project BoB (Agentschap Innoveren en Ondernemen project nr. HBC.2021.0658).

References

[1] E. Iglesias, S. Jozashoori, M.-E. Vidal, Scaling up knowledge graph creation to large and heterogeneous data sources, Journal of Web Semantics (2023).

[2] Sitt Min Oo, G. Haesendonck, B. De Meester, A. Dimou, RMLStreamer-SISO: An RDF Stream Generator from Streaming Heterogeneous Data, in: The Semantic Web – ISWC, 2022.

[3] M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: The Semantic Web – ISWC, 2017.

[4] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: LDOW, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[5] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL, ACM Trans. Database Syst. (2009).

[6] A. Iglesias-Molina, A. Cimmino, E. Ruckhaus, D. Chaves-Fraga, R. García-Castro, O. Corcho, An ontological approach for representing declarative mapping languages, Semantic Web Journal (2022).

[7] A. Iglesias-Molina, D. Chaves-Fraga, F. Priyatna, O. Corcho, Towards the definition of a language-independent mapping template for knowledge graph creation, in: Third International Workshop on Capturing Scientific Knowledge (SciKnow19), 2019.