FunMap: Efficient Execution of Functional Mappings for Knowledge Graph Creation

Samaneh Jozashoori[1(✉)], David Chaves-Fraga[2], Enrique Iglesias[1,3], Maria-Esther Vidal[1], and Oscar Corcho[2]

[1] TIB Leibniz Information Center for Science and Technology and L3S, Hannover, Germany
{samaneh.jozashoori,maria.vidal}@tib.eu
[2] Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain
{dchaves,ocorcho}@fi.upm.es
[3] University of Bonn, Bonn, Germany
s6enigle@uni-bonn.de

Abstract. Data has exponentially grown in the last years, and knowledge graphs constitute powerful formalisms to integrate a myriad of existing data sources. Transformation functions – specified with function-based mapping languages like FunUL and RML+FnO – can be applied to overcome interoperability issues across heterogeneous data sources. However, the absence of engines to efficiently execute these mapping languages hinders their global adoption. We propose FunMap, an interpreter of function-based mapping languages; it relies on a set of lossless rewriting rules to push down and materialize the execution of functions in initial steps of knowledge graph creation. Although applicable to any function-based mapping language that supports joins between mapping rules, FunMap feasibility is shown on RML+FnO. FunMap reduces data redundancy, e.g., duplicates and unused attributes, and converts RML+FnO mappings into a set of equivalent rules executable on RML-compliant engines. We evaluate FunMap performance over real-world testbeds from the biomedical domain. The results indicate that FunMap reduces the execution time of RML-compliant engines by up to a factor of 18, furnishing, thus, a scalable solution for knowledge graph creation.

Keywords: Knowledge graph creation · Mapping rules · Functions

1 Introduction

Knowledge graphs (KGs) have gained momentum due to the explosion of available data and the demand for expressive formalisms to integrate factual knowledge spread across various data sources [14]. KG creation requires the description of schema alignments among data sources and an ontology, as well as the specification of methods to curate and transform data collected from the input sources into a unified format. A rich spectrum of mapping languages has been proposed to specify schema-ontology alignments across data sources implemented in a

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12506, pp. 276–293, 2020.
https://doi.org/10.1007/978-3-030-62419-4_16

variety of semi-structured and structured formats; exemplar approaches include R2RML [6], RML [10], and xR2RML [21]. Furthermore, function-based mapping languages [7,8,17,26] are equipped with abstractions that enable interoperable and reusable specifications of data transformations by means of user-defined functions. Moreover, formalisms like RML+FnO [7] combine the Function ontology and RML, enabling declarative specification of the schema-ontology alignments and data transformations that define the process of KG creation. Albeit expressive, existing mapping languages lack efficient interpreters able to scale up to complex KG creation scenarios. The incoming data avalanche urges KG creation approaches capable of integrating large and diverse data, and efficiently transforming this data to comply with application-specific KG formats.

Problem and Objectives: We tackle the problem of scaled-up KG creation from functional mapping rules and study the impact of functions when applied to large data sources with a high data duplication rate. A KG creation process is defined as a data integration system [19]. Mappings among data sources and the system ontology are expressed using the RDF mapping language (RML) [7] and the Function Ontology (FnO); they define how the ontology concepts are populated with data from the sources in the resulting KG. We aim at transforming complex data integration systems composed of large data sources and mappings with functions into equivalent ones that generates the same KG but in less time.

Our Proposed Approach: We present FunMap, an interpreter of RML+FnO, that converts a data integration system defined using RML+FnO into an equivalent one where RML mappings are function-free. FunMap resembles existing mapping translation proposals (e.g., [2,5,17]) and empowers a KG creation process with optimization techniques to reduce execution time. Transformations of data sources include the projection of the attributes used in the RML+FnO mappings. They are supported on well-known properties of the relational algebra, e.g., the pushing down of projections and selections into the data sources, and enable not only the reduction of the size of data sources but also the elimination of duplicates. Additionally, FunMap materializes functions –expressed in FnO– and represents the results as data sources of the generated data integration system; the translation of RML+FnO into RML mappings that integrate the materialization of functions is performed using joins between the generated RML mappings. The combination of data source and function transformations results in data integration systems where only the data required to execute the RML mappings are retained. The computation of the functions used in the original data integration system is performed once. As a result, the new data integration system's execution is sped up while the same knowledge graph is generated.

Contributions. i) FunMap, an interpreter of RML+FnO that resorts to syntax-based translation [1] to push down projections and selections, and materialize functions. **ii)** Empirical evaluations of the performance of FunMap in real-world testbeds with data of various formats (CSV and Relational), sizes, and degrees of duplication that show reductions in KG creation time by up to a factor of 18.

The remainder of this paper is structured as follows: Section 2 motivates our work using a use case from the biomedical domain. Section 3 describes the set of rewriting and optimization rules that assemble FunMap, while the experimental results are reported in Sect. 4. Finally, Sect. 5 presents the related work and Sect. 6 outlines the main conclusions of the paper and future lines of work.

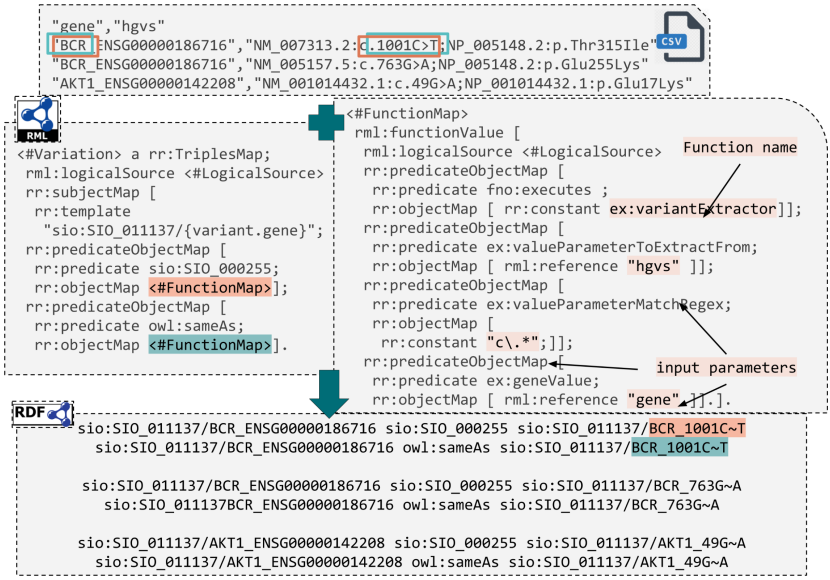


Fig. 1. Motivating example. Knowledge graph construction using RML+FnO mapping rules for the biomedical domain. The input source in the top is transformed to RDF output (at the bottom) through the processing of the mapping (middle) where the transformation functions are defined. Repeated computations of a function negatively impacts on the performance of an RML engine.

2 Preliminaries and Motivating Example

2.1 Preliminaries

The RDF Mapping Language (RML) extends the W3C-standard mapping language R2RML with logical sources (a.k.a. `logicalSource`) in heterogeneous formats (e.g., CSV, Relational, JSON, and XML). As the W3C-standard R2RML, `TriplesMap` corresponds to mapping rules where the resources (a.k.a. `subjectMap`) of an RDF class and their properties (a.k.a. `predicateMap`) are assigned to values (a.k.a. `objectMap`) based on logical data sources. An `objectMap` can be also defined as a reference or a join with the `subjectMap` in another `TriplesMap` (a.k.a. `RefObjectMap` and `joinCondition`, respectively). `subjectMap`, `predicateMap`, and `objectMap` are also referred as `TermMap` in general; they generate RDF terms. FnO is an ontology for describing transformation functions

declaratively; FnO and RML relationship is described in [7]. Accordingly, the `FunctionMap` class is introduced in RML; it defines transformation functions in any part of the `TriplesMap` (`subjectMap`, `predicateMap`, or `objectMap`). These concepts are illustrated in the next example and highlighted in Fig. 1.

2.2 A Real-World Example from the Biomedical Domain

Our work is motivated by the challenges revealed during genomic variant reconciliation while creating a biomedical knowledge graph. Although the vast majority of the single variations in the genome of a person causes no disease, benign variants can appear in sequenced genomic data repeatedly. In addition to the large heterogeneous volumes generated during genome sequencing and analysis, high-frequency of genomic variants impose data integration challenges while collecting genomic data from different sources. Additionally, genomic variants are expressed in diverse standard formats [9] and reported at DNA, RNA, or protein level. Moreover, this representation can be done according to any of the accepted terminologies and genomic reference versions. Unified representations for variants are required to semantically recognize and integrate equivalent variants residing in different data sources. Variant representations can result from a composition of several factors, such as gene name, genomic position, and residue alteration. Pre-processing functions (e.g., FnO functions) are needed to extract and compose values from different attributes from each data source and generate such a combined representation of variants. These functions are part of the data integration system's mapping rules that define the KG creation process.

Figure 1 depicts a mapping rule in RML+FnO where the `FunctionMap` class is utilized. Consider that according to the `LogicalSource` provided in this example, a `FunctionMap` is defined in the mapping rules to create a unified representation for a variant by extracting the values of "gene name" (e.g., BCR) from the attribute `gene` and "coding alteration" (e.g., c.1001C>T) from the attribute named `hgvs` and combine them (e.g., BCR_1001C~T). Current approaches evaluate `FunctionMap` for each `variant`, which can be expensive in presence of large data sources. Nevertheless, the large number of redundant values leaves room for the scalable transformations to execute functional mappings.

3 The FunMap Approach

FunMap is an interpreter of data integration systems $DIS_G = \langle O, S, M \rangle$, where O stands for a unified ontology, and S and M represent sets of sources and mapping rules, respectively [19]. The evaluation of DIS_G (a.k.a. $RDFize(DIS_G)$) results into a knowledge graph G that integrates data from S according to the mapping rules in M; entities and properties in G are described in terms of O. A complex data integration system DIS_G consists of large data sources with high-duplicated data and mapping rules including functions for both schema-ontology alignments and data transformations. FunMap converts DIS_G into an

Table 1. Summary of the notation used for defining FunMap

Notation	Explanation
$DIS_G = \langle O, S, M \rangle$	Data Integration System which creates a KG G
O	Unified Ontology of $DIS_G = \langle O, S, M \rangle$
S	Finite set of Data Sources S_i of $DIS_G = \langle O, S, M \rangle$
M	Finite set of `TriplesMap`s T_i in $DIS_G = \langle O, S, M \rangle$
$RDFize(.)$	A function producing RDF triples from a data integration system
T'_i and T'_k	`TriplesMap`s resulting of applying MTRs
F_i	A Transformation Function in a `TriplesMap` in M
S'	Finite set of Data Sources S'_i resulting of applying DTRs
M'	Finite set of Mapping Rules M'_i resulting of applying MTRs
S_i^{output}	Data source resulting of applying DTR1, with attributes o'_i and a'_i representing the materialization of a transformation function F_i
$S_i^{project}$	Data source resulting of applying DTR2

equivalent data integration system that creates the same knowledge graph but in less time. Table 1 summarizes the notation utilized in the FunMap approach.

Problem Statement: Given a data integration system $DIS_G = \langle O, S, M \rangle$, the problem of scaled-up knowledge graph creation from functional mappings requires the generation of a data integration system $DIS'_G = \langle O, S', M' \rangle$:

- The knowledge graphs resulting of the evaluations of both data integration systems are the same, i.e., $RDFize(DIS'_G = \langle O, S', M' \rangle) = RDFize(DIS_G = \langle O, S, M \rangle)$ where $RDFize(.)$ is a function producing RDF triples utilizing the input data integration system.
- The execution time of $RDFize(DIS'_G = \langle O, S', M' \rangle)$ is *less than* the execution time of $RDFize(DIS_G = \langle O, S, M \rangle)$.

Solution: FunMap implements a heuristic-based approach; it relies on the assumption that eliminating duplicates, maintaining in the data sources only the attributes mentioned in the mappings, and materializing the functions in the mappings, reduces the execution time of knowledge graph creation process. FunMap receives a data integration system $DIS_G = \langle O, S, M \rangle$ where the mappings M are expressed in RML+FnO. FunMap interprets the mappings in M and converts DIS into the data integration system DIS'_G in which the mappings M' are function free and duplicates in the data sources S' are reduced. Figure 2 depicts the FunMap approach; it performs a syntax-based translation of the mappings in M and ensures that each redundant function is evaluated

exactly once on the same data values. FunMap transforms S to S' by means of data transformation rules (DTR1 and DTR2). For each F_i over a given S_i, DTR1 creates a temporal source S'_i that includes the attributes from S_i that correspond to the input of F_i; it also generates a source S_i^{output} that contains the attributes in S'_i and attributes representing the output of F_i. For each `FunctionMap` defined over a source S_i, DTR2 creates a source $S_i^{project}$ that includes all attributes of S_i used in the `FunctionMap`. Additionally, FunMap converts mapping rules that include functions by using mapping transformation rules (MTRs); a `FunctionMap` is transformed into `FunctionMaps` without functions while connected by `joinConditions`; initially, S' and S are equal, as well as M' and M. Properties 1, 2, and 3 state the pre- and post-conditions of DTRs and MTRs.

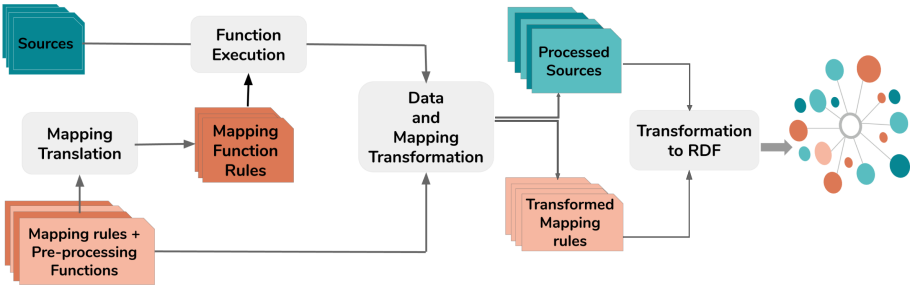


Fig. 2. The FunMap approach

3.1 Transformation Rules in FunMap

The FunMap syntax-based translation component parses `FunctionMaps` exactly once, i.e., `FunctionMaps` repeated in various mappings are not evaluated more than once. Given `FunctionMaps`, original data sources, and mappings, FunMap executes transformation rules on data sources and mappings, accordingly. Meanwhile, given the transformed data sources, FunMap detects that a `FunctionMap` has been computed for a given value and avoids repeating this computation. As an outcome, FunMap provides **a)** a new set of data sources S' consisting of the original ones in conjunction with transformed data sources, and **b)** a set M' of transformed function-free mappings. FunMap is loyal to the formats of data sources and mappings. Thus, any RDF mapping language is compatible with the process implemented in FunMap, as far as the language enables the definition of joins between mapping rules. Next, we present the transformation rules.

Data Source Transformation Rules (DTRs): Considering the fact that a `TriplesMap` may only be used some attributes of a dataset, FunMap relies on the properties of the relational algebra and performs DTRs to project only the attributes mentioned in the `TriplesMap`. DTRs are followed by transformation rules (MTRs) that update mappings defined over the transformed data sources.

Fig. 3. Example of DTR and Object-based MTR. On the left, an exemplary mapping including two TriplesMaps and a FunctionMap provided by the original data integration system. On the right side, the mappings are transformed by FunMap including two new TriplesMaps and one new TriplesMap.

DTR1: Projection of Functional Attributes: For each transformation function F_i over a given source S_i in the set of data sources S, FunMap projects all attributes a'_i in S_i that are input attributes of F_i, into a temporal data source S'_i followed by duplicate removal. Subsequently, it evaluates F_i over S'_i and stores the results into the attribute o_i. Lastly, it creates a new data source S_i^{output} with the attributes a'_i and o_i; S_i^{output} is added to S'.

DTR2: Projection of Non-functional Attributes: FunMap provides an additional DTR to further optimize the knowledge graph creation process. Exploiting transformation rules that are proposed in [16], FunMap projects all attributes in S_i that are needed by `TriplesMap` including those that are received by `FunctionMap` as input into a new data source $S_i^{project}$ which is added to S'. To better conceive DTRs, consider the original mappings in Fig. 3 (left-side) and corresponding data source `source1.csv` that can be seen in Fig. 4. As shown in Fig. 3, `FunctionMap1` receives `Mutation genome position` as input. According to DTR1, FunMap projects `Mutation genome position` from `source1` into a new data source named `output1.csv` which is shown in Fig. 5c. The rows number

ID	Gene name	GRCh	Mutation genome position	Mutation CDS	Primary site	GENOMIC_MUTATION_ID	...
1	DGCR6L	37	22:20302597-20302597	c.514-250C>A	Liver	COSV50619134	...
2	HMCN1	37	1:186072702-186072702	c.10672C>T	lung	COSV54901969	...
3	SLC5A10_ET0 000039564	37	17:18874996-18874996	c.597+465T>A	Skin	COSV58755801	...
4	HMCN1_ET000 00367492	37	1:186072702-186072702	c.10672C>T	Skin	COSV54901969	...
5	COL21A1_ET0 000037081	37	6:56246049-56246049	c.-39+12720G>A	Prostate	COSV63690608	...
6	AKT3	37	1:243692781-243692781	c.1251+16031C>G	Pancreas	COSV55606438	...
7	WDFY4_ET000 00413659	37	10:50044166-50044166	c.*1825+3412G>A	Oesophagus	COSV55433638	...
...	...		...	...	...	...	...

Fig. 4. Original data source for KG creation. The data source includes many attributes among which only a few are required by the transformation function or function-free mappings in the process of knowledge graph creation.

2 and 4 have the same value for attribute `Mutation genome posit-ion` which leads FunMap to remove the duplicated value from `output1.csv`. Afterwards, `FunctionMap1` is evaluated given `output1.csv` as input and the output values are inserted as a new attribute named `functionOutput` into the `output1.csv` data source. Moreover, attributes `GENOMIC_MUTATION_ID` and `Primary site` from `source1.csv` that are in `TriplesMap1` are projected into the new data source that is shown in Fig. 5a and duplicated values are removed. Similarly, `Projected2.csv` is created based on the attributes of `TriplesMap2`.

Mapping Transformation Rules (MTRs). Mappings are transformed to create the same knowledge graph utilizing the transformed data sources. MTRs are defined considering the role of a transformation function F_i in each `TriplesMap` T_i. **I)** F_i as an `ObjectMap`: We refer to the MTRs that are required in this case as `Object-based`. First of all, for each F_i, a new `TriplesMap` T_i' is created; it refers to the data source generated as the outcome of F_i, i.e., S_i^{output}. Accordingly, the `SubjectMap` of T_i' refers to the output attributes o_i in S_i^{output}. Afterwards, in `TriplesMap` T_i where F_i is presented as an `ObjectMap`, F_i is replaced by a `joinCondition` which joins T_i and T_i' over attributes a_i', i.e., the input attributes of F_i. Moreover, the `logicalSource` of T_i is changed to $S_i^{project}$, i.e., the corresponding projected data source provided as an outcome of DTR2. **II)** F_i as a `SubjectMap`: Contrary to the `Object-based`, in this set of MTR - we refer to as `Subject-based`- for each `predicateObjectMap` that follows a F_i of the type `SubjectMap`, a new `TriplesMap` T_i' refers to the data source $S_i^{project}$ which is generated as an outcome of DTR2 by projecting the attribute a_i' from S_i that are referenced as `objectMap` in the original `predicateObjectMap`. The `subjectMap` of T_k' –the transformed T_i – refers to the o_i and its `logicalSource` is S_i^{output}. Note that `subjectMap` of T_i' is by definition a `TermMap`, which means that its value can be any RDF term according to the RML specification. Each `objectMap` in T_i that is a `FunctionMap` is replaced by a `joinCondition` between

ID	Mutation genome position	GENOMIC_MUTATION_ID
1	22:20302597-20302597	COSV50619134
3	17:18874996-18874996	COSV58755801
4	1:186072702-186072702	COSV54901969
5	6:56246049-56246049	COSV63690608
6	1:243692781-243692781	COSV55606438
7	10:50044166-50044166	COSV55433638
...	...	...

(a) Projected1

ID	Mutation genome position	Gene name
1	22:20302597-20302597	DGCR6L
2	1:186072702-186072702	HMCN1
3	17:18874996-18874996	SLC5A10_ET0000039564
4	1:186072702-186072702	HMCN1_ET00000367492
5	6:56246049-56246049	COL21A1_ET0000037081
6	1:243692781-243692781	AKT3
7	10:50044166-50044166	WDFY4_ET00000413659
...	...	...

(b) Projected2

ID	Mutation genome position	functionOutput
1	22:20302597-20302597	22:20302597:20302597
3	17:18874996-18874996	17:18874996:18874996
4	1:186072702-186072702	1:186072702:186072702
5	6:56246049-56246049	6:56246049:56246049
6	1:243692781-243692781	1:243692781:243692781
7	10:50044166-50044166	10:50044166:50044166
...	...	...

(c) Output1

Fig. 5. Transformed sources generated by FunMap. The DTR2 generates a new source by projecting attributes for each TripleMap (a and b) while DTR1 projects input and output attributes of each FunctionMap into a new source (c). Both remove the generated duplicates.

T_i and corresponding T_i' over input attributes a_i' of F_i. In both cases, the transformed T_i– denoted as T_k'– and T_i' are added to M' and T_i is removed from M'. Figures 3 and 6 illustrate two examples of rewritten mappings based on DTRs and MTRs. In the left side of both figures, the original mappings are presented while the transformed mappings are depicted on the right side. In the transformed mappings in Fig. 3, `TriplesMap3` is created for `FunctionMap1`; it refers to the attribute `functionOutput` in the projected data source `output1.csv`– shown in Fig. 5c. Then, `FunctionMap1` is replaced in both `TriplesMap1` and `TriplesMap2` by a join condition over the attribute `Mutation genome position` which is the input attribute of `FunctionMap` in the original mapping file as it is highlighted by the same **color**. Accordingly, data sources -highlighted- of `TriplesMaps` are also transformed to refer to the projected data sources. Consider Fig. 6 where `FunctionMap` is a `subjectMap`. In both `predicateObjectMa-ps` of `TriplesMap1`, `FunctionMap1` is replaced by a `joinCondition` over the attribute `Mutation genome position` that is the input of `FunctionMap1`. To better clarify the performed transformation, consider the first `predicateObjectMap` in `TripleMap1` in the original mappings; the `predicate` is **represents** and the `ObjectMap` refers to the attribute **Mutation**. After the transformation, the first `predicateObjectMap` has the same `predicate` **represents** and through the `joinCondition` refers to the same attribute **Mutation** in `projected1.csv`.

Pre- and post-conditions of Data Source Transformation Rules (DTRs) and Mapping Transformation Rules (MTRs) are stated in the following properties:

Property 1 (Lossless Function). Given data integration systems $DIS_G = \langle O, S, M \rangle$ and $DIS_G' = \langle O, S', M \rangle$ such that DIS_G' is the result of applying

Fig. 6. Example of Subject-based MTR. An example of mappings including a TriplesMaps and FunctionMap are illustrated on the left and their transformed version including three TriplesMap are shown on the right side.

one DTR1 transformation to DIS_G. Then, there are data sources S_i and S_i^{output} in S and S', respectively, and the following statements hold:

- $S' - S = \{S_i^{output}\}$, there is a mapping T_i in M with a function F_i, and $Attrs$ contains the attributes a_i' of F_i in S_i and the output attributes o_i of F_i.
- S_i^{output} comprises the attributes $Attrs$ and $\pi_{a_i'}(S_i^{output}) = \pi_{a_i'}(S_i)$.
- For each tuple $t_{i,j}$ in S_i^{output}, the values of the attributes o_i in $t_{i,j}$ correspond to the result of F_i over the values of a_i' in $t_{i,j}$, i.e., $t_{i,j}.o_i = F_i(t_{i,j}.a_i')$.

Property 2 (Lossless Projection). Given data integration systems $DIS_G = \langle O, S, M\rangle$ and $DIS_G' = \langle O, S', M\rangle$ such that DIS_G' is the result of applying one DTR2 transformation to DIS_G. Then, there are data sources S_i and $S_i^{project}$ in S and S', respectively, and the following statements hold:

- $S' - S = \{S_i^{project}\}$, and there is a mapping T_i in M defined over the attributes $Attrs$ from S_i, and $S_i^{project} = \pi_{Attrs}(S_i)$.

Property 3 (Lossless Schema-Ontology Alignments).[1] Given data integration systems $DIS_G = \langle O, S, M\rangle$ and $DIS_G' = \langle O, S, M'\rangle$ such that DIS_G' is the result of applying one MTR transformation to DIS_G. Then, there are `TriplesMaps` T_i in M, and T_i' and T_k' in M', and the following statements hold:

- $M - M' = \{T_i\}$ and $M' - M = \{T_i', T_k'\}$.

[1] Similarly, this property can be stated for the result of applying MTR over the subject position of a property in a mapping of a data integration system.

- There is a function F_i in T_i as the `ObjectMap` of a `PredicateMap` p, and there is a data source S_i^{output} in S which is the `LogicalSource` of T_i. The attributes of S_i^{output} are the union of a'_i and o_i, while a'_i and o_i are input and output attributes of F_i, respectively.
- T_i and T'_k are defined over the same `LogicalSource` $S_i^{project}$. S_i^{output} is the `LogicalSource` of T'_i and o_i is the `SubjectMap` of T'_i.
- T_i and T'_k only differ on the `ObjectMap` p. In T_i, `ObjectMap` of p is defined as F_i, while in T'_k, a `joinCondition` to T'_i on a'_i defines the `ObjectMap` of p.

4 Experimental Evaluation

We evaluate FunMap[2] in comparison to current approaches that create a knowledge graph using the specified data sources and RML+FnO mappings. We aim to answer the following research questions: **Q1)** What is the impact of data duplication rate in the execution time of a knowledge graph creation approach? **Q2)** What is the impact of different types of complexity over transformation functions during a knowledge graph creation process? **Q3)** How does the repetition of a same function in different mappings affect the existing RML engines? **Q4)** What is the impact of relational data sources in the knowledge graph creation process? All the resources used to perform this evaluation are available in our Github repository[3]. The experimental configuration is as follows:

Datasets and Mappings. To the best of our knowledge, there are no testbeds to evaluate the performance of a knowledge graph construction approach that applies functional mappings. Consequently, following the real-world scenario that initially motivated this research, we create our testbed from the biomedical domain. We generate a baseline dataset by randomly selecting 20,000 records from the coding point mutation dataset in COSMIC[4] database. We keep all 39 attributes of the original dataset in the baseline dataset, while only five to seven of them are utilized in mappings. In total, four different mapping files are generated consisting of one `FunctionMap` and four, six, eight, or ten `TriplesMap`s with a `predicateObjectMap` linked to the function. To additionally validate FunMap in case of large-sized data, we create another dataset following the same criteria, with 4,000,000 records and the size of about 1.3 GB.

Engines. The baselines of our study are three different open source RML-complaint engines that are able to execute RML+FnO mappings and have been extensively utilized in multiple applications and tested by the community: SDM-RDFizer v3.0 [15], RMLMapper[5] v4.7, and RocketRML[6] v1.1.[7]. In order to evaluate the impact of transformation rules, we implement FunMap v1.0 on the top

[2] https://doi.org/10.5281/zenodo.3993657.
[3] https://github.com/SDM-TIB/FunMap.
[4] https://cancer.sanger.ac.uk/cosmic GRCh37, version90, released August 2019.
[5] https://github.com/RMLio/rmlmapper-java.
[6] https://github.com/semantifyit/RocketRML/.
[7] We name them SDM-RDFizer**(RML+FnO), RMLMapper**(RML+FnO), and RocketRML**(RML+FnO).

of the aforementioned engines with DTR2 optimization as an optional parameter. We refer to the approach which applies FunMap excluding DTR2 as FunMap$^-$[8]. We created a docker image per tested engine for reproducibility.

Metrics. *Execution time:* Elapsed time spent by an engine to complete the creation of a knowledge graph and also counts FunMap pre-processing; it is measured as the absolute wall-clock system time as reported by the `time` command of the Linux operating system. Each experiment was executed five times and average is reported. The experiments were executed on an Ubuntu 16.04 machine with Intel(R) Xeon(R) Platinum 8160, CPU 2.10 GHz and 700 Gb RAM.

Experimental Setups. Based on our research questions, we set up in overall 198 experiments as the combinations of the following scenarios. We create two datasets from our baseline with 25% and 75% duplicates which means in the 25% duplicate dataset, 25% and in the 75% duplicate dataset, 75% of the records are duplicated. Additionally, two functions with different levels of complexity are created. We describe the complexity level of the functions based on the number of required input attributes and operations to be performed. Accordingly, "simple" function is defined to receive one input attribute and perform one operation, while a "complex" function receives two input attributes and completes five operations. In total, we create eight mapping files including four, six, eight, and ten `TriplesMap` and one `FunctionMap` to be either "simple" or "complex". Additionally, six experiments using 75% duplicate datasets of 20,000 and 4,000,000 records and a mapping file including ten complex functions are set up in order to be run over a relational database (RDB) implemented in MySQL 8.0[9].

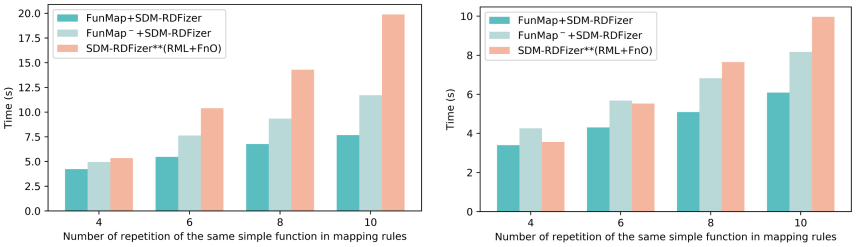
4.1 Discussion of Observed Results

In this section, we describe the outcomes of our experimental evaluation. Figure 7 reports on the execution time of the different testbeds in which the functions are considered to be "simple" whereas Fig. 8 shows the experiments involving "complex" functions. Both figures represent the total execution time for constructing the knowledge graph applying selected engines (i.e., SDM-RDFizer, RMLMapper, and RocketRML) in three different configurations: a) the current version of the engine that is able to directly interpret RML+FnO mappings in the engine (e.g., RMLMapper**(RML+FnO)); b) FunMap$^-$ in conjunction with the engine (e.g., FunMap$^-$+RMLMapper); and c) FunMap together with the engine (e.g., FunMap+RMLMapper). In the case of all the configuration of RocketRML, we only provide the results for the execution of simple functions because the engine does not execute joins with multiple conditions[10] correctly,

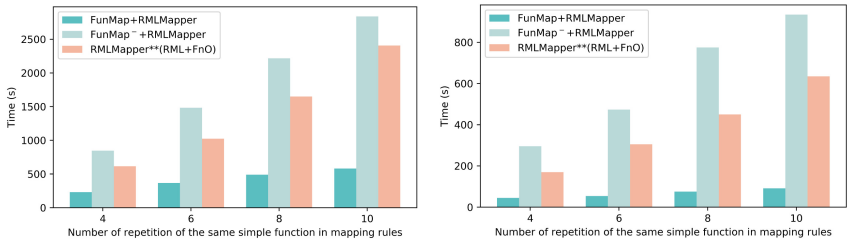
[8] We name these combined engines as follows: a) FunMap: FunMap+SDM-RDFizer, FunMap+RMLMapper, and FunMap+RocketRML; b) FunMap$^-$: FunMap$^-$+SDM-RDFizer, FunMap$^-$+RMLMapper, and FunMap$^-$+RocketRML.

[9] https://www.mysql.com/.

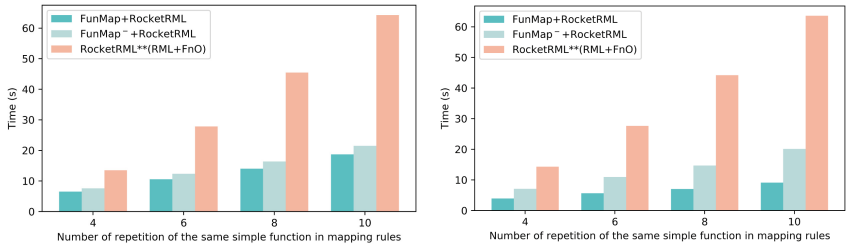
[10] Check an example in the zip file of the supplementary material.



(a) SDM-RDFizer - 25% of duplicates (b) SDM-RDFizer - 75% of duplicates



(c) RMLMapper - 25% of duplicates (d) RMLMapper - 75% of duplicates

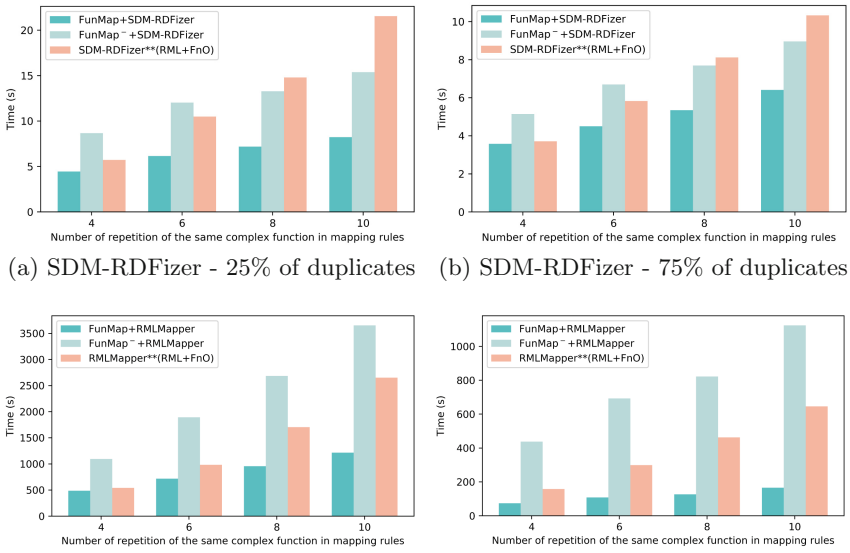


(e) RocketRML - 25% of duplicates (f) RocketRML - 75% of duplicates

Fig. 7. Total execution time of experiments with simple functions 25–75% of duplicates. SDM-RDFizer, RMLMapper and RocketRML executing simple functions in RML+FnO mappings and with FunMap and FunMap$^-$.

hence, the proposed optimizations cannot be applied. For the rest of the experiments, we have verified that the results are the same for all the approaches in terms of cardinality and correctness. The results obtained by the application of SDM-RDFizer with the repetition of simple functions (Figs. 7a and 7b) reflect an improvement of the execution time when FunMap is applied in the process. With the growth of number of duplicates and repeated functions, the difference between the performance of SDM-RDFizer**(RML+FnO) and FunMap+SDM-RDFizer increases. Using this engine, FunMap$^-$ shows the same behavior as FunMap, however, in the case of having a large number of duplicates and a few repeated functions FunMap$^-$ does not improve the performance of SDM-RDFizer**(RML+FnO). In the case of using RMLMapper (Figs. 7c and 7d), we

observe that the results obtained together with FunMap$^-$ (i.e., DTR1 optimization) do not show better performance than RMLMapper**(RML+FnO). DTR1 which only focuses on transforming functions, delegates the removal of the duplicates to the engine which is not accomplished efficiently by RMLMapper. However, in FunMap+RMLMapper, that includes DTR1 and DTR2 optimizations, duplicates are removed before the execution of the RML mappings and leads to obtain the results that clearly show improvements with respect to the baseline. In the same manner as the SDM-RDFizer, the number of repetitions of the functions affects the execution time of the RMLMapper**(RML+FnO), while FunMap maintains similar execution times. Finally, RocketRML (Figs. 7e and 7f) seems not to be affected by the number of duplicates over the input data, obtaining similar execution times for 25% and 75% rate for RocketRML**(RML+FnO). However, the number of repetitions over functions impacts the performance of RocketRML**(RML+FnO), increasing the total execution time. The incorporation of DTR1 (i.e., FunMap$^-$+RocketRML) and DTR2 (i.e., FunMap+RocketRML) enhances the performance and scalability during the construction of the knowledge graph, obtaining a similar behavior as the other two tested engines.



(a) SDM-RDFizer - 25% of duplicates (b) SDM-RDFizer - 75% of duplicates

(c) RMLMapper - 25% of duplicates (d) RMLMapper - 75% of duplicates

Fig. 8. Total execution time for complex functions 25–75% of duplicates. SDM-RDFizer and RMLMapper executing complex functions in RML+FnO mappings and with FunMap and FunMap$^-$.

The effect of function complexity over SDM-RDFizer can be observed in Figs. 8a and 8b. Whenever the number of repetitions is low (4–6), the join with multiple conditions affects FunMap$^-$+SDM-RDFizer, obtaining worse results

than SDM-RDFizer**(RML+FnO). However, if repetitions increase (8–10), DTR1 empowers SDM-RDFizer**(RML+FnO) due the reduction of repeated operations during the evaluation of the mappings. Conversely, FunMap+SDM-RDFizer exhibits better results than SDM-RDFizer**(RML+FnO) in all the testbeds. Finally, the behavior of RMLMapper – when it has to execute complex transformation functions (Figs. 8c and 8d) – is affected in terms of execution time for the configuration FunMap^{-}+RMLMapper in comparison to the case of simple functions. As similar as SDM-RDFizer, the join with several conditions is impacting the performance. However, together with data transformation optimizations, FunMap+RMLMapper outperforms the baseline.

The experimental results on RDBs show even more significant improvement in the performance of both RMLMapper and SDM-RDFizer in the presence of FunMap. In FunMap+RMLMapper, applying `joins` in the SQL queries that define the `logicalSources` instead of using `joinConditions` reduces execution time by up to a factor of 18. These results evidence that `joinConditions` are not efficiently implemented by RMLMapper, and explain why FunMap+RMLMapper is showing less improvement compared to FunMap+SDM-RDFizer in Fig. 8. Moreover, FunMap+SDM-RDFizer successfully performs on the large-sized relational dataset of 1.3 GB in 5,670.67 s, while SDM-RDFizer**(RML+FnO) cannot create the KG and times out after 10,000 s.

In overall, we observe that the configurations that interpret RML+FnO mappings directly are affected by the repetition of the functions and the degree of data duplicates, i.e., execution time monotonically increases with number of functions and data duplication degree. In contrast, the incorporation of FunMap to the engines shows less fluctuated behavior when the data duplication rate increases. Additionally, the studied engines handle the repetition of the functions during the construction of the knowledge graph thanks to the pushing down of the execution of the functions directly over the dataset. In summary, the observed results indicate that the FunMap heuristics improve the performance of data integration systems and generate solutions to the problem of scaled-up knowledge graph construction. The effectiveness of the proposed transformations has been empirically demonstrated on various RML+FnO and RML-compliant engines. However, we observe that there are cases where the application of DTR1 alone is not enough (i.e., FunMap^{-}), being required the applications of all the transformations (i.e., DTRs and MTRs) to provide an effective solution.

5 Related Work

Solutions provided to the problem of KG creation from (semi)-structured data are gaining momentum in practitioners and users [5]. The seminal paper by Lenzerini [19] provides a formal framework for solving this problem and represents a pivot for Ontology-Based Data Access/Integration (OBDA/I) [22] and the corresponding optimization approaches. Gawriljuk et al. [12] present a framework for incremental knowledge graph creation. While optimized SPARQL-to-SQL query translation techniques [4] are implemented to support virtual knowledge

graph creation [3,23]. Albeit efficient, these approaches do not support data transformations (i.e., functions), preventing an efficient evaluation of declarative knowledge graph creation processes. Although FunMap is focused on customized transformation functions for materialized KGs, it can be applied on top of these OBDA approaches. However, this would require the definition of new transformation rules able to push down the corresponding functions into SQL engines.

Rahm and Do [24] have reported the relevance of data transformations expressed with functions during data curation and integration. Grounding on this statement, different approaches have been proposed for facilitating the definition of functions to enhance data curation (e.g., [11,13,25]). Similarly, declarative languages have been proposed to allow for the definition of functions in the mappings; exemplar approaches include R2RML-F [8], FunUL [17], RML+FnO [7], and D-REPR [26]. Moreover, mapping engines enable to interpret functions in declarative mappings (e.g., Squerall [20], RMLStreamer[11] and CARML[12] for RML+FnO), as well as in non-declarative formalisms [18]. FunMap optimizations currently are performed over static data and require the implementation of the `joinCondition` by a KG creation engine. However, RMLStreamer works over streaming data, while CARML does not entirely cover RML joins. Additionally, Squerall is a SPARQL query engine over heterogeneous data able to process RML+FnO on the fly, but Squerall does not implement RML joins. Despite the rich repertory of these approaches, optimizing the declarative description of complex data integration systems remains still open. The absence of frameworks capable of efficiently execute complex data integration systems negatively impacts on the global adoption of existing formalisms in real-world applications of knowledge graphs. FunMap aims at bridging this gap and offering an alternative of evaluating RML+FnO over existing RML-compliant engines.

6 Conclusions and Future Work

We addressed the problem of scaled-up KG creation in complex data integration systems, i.e., systems with large data sources, high data duplication rate, and functional mappings. We presented a heuristic-based approach for efficiently evaluating data integration systems with data sources in diverse formats (e.g., CSV or relational). The proposed heuristics are implemented in FunMap, an interpreter of RML+FnO, that converts data integration systems in RML+FnO into equivalent data integration systems specified in RML. Besides shaping an RML-engine independent interpreter of RML+FnO, FunMap generates data integration systems that enhance RML-compliant engines whenever transformation functions are repeatedly used, and data sources are large and have highly-duplicated data. Empirical evaluations of the combination of FunMap with RML-compliant engines suggest that the execution time of RML+FnO can be reduced by up to a factor of 18. Thus, FunMap widens the repertory of tools to scale up knowledge graphs to the enormous increase of incoming data and

[11] https://github.com/RMLio/RMLStreamer.
[12] https://github.com/carml/carml.

ease the development of real-world KG applications. As the main limitation, FunMap can only be applied with an RML-compliant engine which supports either `joinCondition` or RDB on the backend. We plan to devise cost-based optimization approaches that, together with the proposed heuristics, allow for the generation of the best solution for a complex data integration system in RML+FnO.

Acknowledgments. This work has been partially supported by the EU H2020 project iASiS No. 727658, the ERAMed project P4-LUCAT No. 53000015, Ministerio de Economía, Industria y Competitividad, and EU FEDER funds under the DATOS 4.0: RETOS Y SOLUCIONES - UPM Spanish national project (TIN2016-78011-C4-4-R) and by the FPI grant (BES-2017-082511).

References

1. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers, Principles, Techniques, vol. 7, no, 8. Addison Wesley, Boston (1986)
2. Ali, S.M.F., Wrembel, R.: Towards a cost model to optimize user-defined functions in an ETL workflow based on user-defined performance metrics. In: Advances in Databases and Information Systems, ADBIS (2019)
3. Calvanese, D., et al.: Ontop: answering SPARQL queries over relational databases. Semant. Web **8**(3), 471–487 (2017)
4. Chebotko, A., Lu, S., Fotouhi, F.: Semantics preserving SPARQL-to-SQL translation. Data Knowl. Eng. **68**(10), 973–1000 (2009)
5. Corcho, O., Priyatna, F., Chaves-Fraga, D.: Towards a new generation of ontology based data access. Semant. Web J. **11**(1), 153–160 (2020)
6. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012. W3C (2012)
7. De Meester, B., Maroy, W., Dimou, A., Verborgh, R., Mannens, E.: Declarative data transformations for linked data generation: the case of DBpedia. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) ESWC 2017. LNCS, vol. 10250, pp. 33–48. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58451-5_3
8. Debruyne, C., O'Sullivan, D.: R2RML-F: towards sharing and executing domain logic in R2RML mappings. In: LDOW Workshop (2016)
9. den Dunnen, J.T., et al.: HGVS recommendations for the description of sequence variants: 2016 update. Hum. Mutat. **37**(6), 564–569 (2016)
10. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: a generic language for integrated RDF mappings of heterogeneous data. In: 7th Workshop on Linked Data on the Web (2014)
11. Galhardas, H., Florescu, D., Shasha, D., Simon, E., Saita, C.: Declarative data cleaning: Language, model, and algorithms (2001)
12. Gawriljuk, G., Harth, A., Knoblock, C.A., Szekely, P.: A scalable approach to incrementally building knowledge graphs. In: Fuhr, N., Kovács, L., Risse, T., Nejdl, W. (eds.) TPDL 2016. Lecture Notes in Computer Science, vol. 9819. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43997-6_15
13. Gupta, S., Szekely, P., Knoblock, C.A., Goel, A., Taheriyan, M., Muslea, M.: Karma: a system for mapping structured sources into the semantic web. In: Simperl, E., et al. (eds.) ESWC 2012. LNCS, vol. 7540. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46641-4_40

14. Hogan, A., et al.: Knowledge graphs. CoRR, abs/2003.02320 (2020)
15. Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., Vidal, M.-E.: SDM-RDFizer: an RML interpreter for the efficient creation of RDF knowledge graphs. In ACM International Conference on Information and Knowledge Management, CIKM (2020)
16. Jozashoori, S., Vidal, M.-E.: MapSDI: a scaled-up semantic data integration framework for knowledge graph creation. In: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (eds.) OTM 2019. LNCS, vol. 11877, pp. 58–75. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33246-4_4
17. Junior, A.C., Debruyne, C., Brennan, R., O'Sullivan, D.: FunUL: a method to incorporate functions into uplift mapping languages. In: International Conference on Information Integration and Web-based Applications and Services (2016)
18. Lefrançois, M., Zimmermann, A., Bakerally, N.: Flexible RDF generation from RDF and heterogeneous data sources with SPARQL-generate. In: Ciancarini, P., et al. (eds.) EKAW 2016. LNCS (LNAI), vol. 10180, pp. 131–135. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58694-6_16
19. Lenzerini, M.: Data integration: a theoretical perspective. In: ACM Symposium on Principles of Database Systems (2002)
20. Mami, M.N., Graux, D., Scerri, S., Jabeen, H., Auer, S., Lehmann, J.: Squerall: virtual ontology-based access to heterogeneous and large data sources. In: Ghidini, C., et al. (eds.) ISWC 2019. LNCS, vol. 11779, pp. 229–245. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30796-7_15
21. Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J.: Translation of relational and non-relational databases into RDF with xR2RML. In: WEBIST, pp. 443–454. SciTePress (2015)
22. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 133–173. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77688-8_5
23. Priyatna, F., Corcho, O., Sequeda, J.F.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In: International Conference on World Wide Web, WWW 2014 (2014)
24. Rahm, E., Do, H.H.: Data cleaning: problems and current approaches. IEEE Data Eng. Bull. **23**(4), 3–13 (2000)
25. Raman, V., Hellerstein, J.M.: Potter's wheel: an interactive data cleaning system. In: VLDB, vol. 1 (2001)
26. Vu, B., Pujara, J., Knoblock, C.A.: D-REPR: a language for describing and mapping diversely-structured data sources to RDF. In: International Conference on Knowledge Capture (2019)