A Declarative Formalization of R2RML Using Datalog and Its Efficient Execution

Ali Elhalawati[1], Jan Van den Bussche[2] and Anastasia Dimou[1]

[1]*KU Leuven - Flanders Make@KULeuven - Leuven.AI, Leuven, Belgium*
[2]*Data Science Institute, Universiteit Hasselt, Hasselt, Belgium*

Abstract

R2RML is the W3C-recommended mapping language for defining declarative, customized mappings from relational databases to knowledge graphs, particularly in data integration and schema transformation scenarios. R2RML, like other mapping languages, enables viewing existing relational data in RDF, expressed in a structure and target vocabulary of the mapping author's choice. Despite its broad adoption and plethora of extensions, the complete semantics of R2RML have not been concretely formalized so far. In this paper, we provide a declarative, computable, and rule-based formalization of R2RML through Datalog. We formally define the syntax of R2RML, provide a translation of its semantics into a Datalog program that can be used to evaluate RDF graphs, and discuss the associated complexity. The Datalog program defines output relations for the correct set of triples and quadruples, given any relational data as input relations. We validate the accuracy of our Datalog-based semantics by executing the R2RML test cases using a prototype implementation based on our approach. Our work lays the groundwork for further investigation into the properties and extensions of R2RML, unlocks the various benefits of Datalog reasoning in RDF generation, and introduces a promising approach for generating RDF graphs using any out-of-the-box Datalog reasoner.

Keywords

R2RML, Datalog, Knowledge Graph Construction, Reasoning

1. Introduction

Over the years, Ontology-Based Data Access (*OBDA*) systems have proved their value in both industry and academia [1] in complex data integration environments by providing access to raw data through a conceptual layer in the form of a knowledge graph (*KG*). An OBDA system structures data access through an ontology [1], which is a formal structure that represents classes, properties, and constraints within a domain. Mappings define the relationship between the data and the ontology by constructing the knowledge graph from the data following the ontology. Mapping languages define how to generate KGs from diverse data sources by providing the mappings to an OBDA system. If the data sources are relational databases (*RDB*s), an OBDA system has three main components: the relational data, the ontology, and the mappings that construct the KG from the relational data following the ontology.

While RDF graphs are the widely adopted representation for building KGs, in 2012, the RDB2RDF W3C Working Group published two W3C recommendations for mapping relational data to RDF graphs: a direct mapping [2] and a language for defining customized mappings [3], the RDB to RDF Mapping Language (R2RML) [4]. In the *direct mapping* of a database, the structure of the resulting RDF graph directly reflects the database structure, and the target RDF vocabulary directly reflects the names of database schema elements. To the contrary, in the *customized mapping* of a database, mapping authors define customized views over the relational data, expressed in a structure and target vocabulary of their choice. R2RML mappings define how each schema element is represented as RDF triples (subject, predicate, object). An R2RML processor then executes these mappings to construct RDF graphs.

Although the two W3C recommendations were published together, only the formal semantics of

RuleML+RR'25: Companion Proceedings of the 9th International Joint Conference on Rules and Reasoning, September 22–24, 2025, Istanbul, Turkiye

ali.elhalawati@kuleuven.be (A. Elhalawati); jan.vandenbussche@uhasselt.be (J. Van den Bussche); anastasia.dimou@kuleuven.be (A. Dimou)
0000-0003-1457-0031 (A. Elhalawati); 0000-0003-0072-3252 (J. Van den Bussche); 0000-0003-2138-7972 (A. Dimou)

direct mapping are concretely defined [5]. No detailed formalization of R2RML exists so far, despite the proposal of several alternatives and extensions over R2RML [3]. There are works formalizing partial fragments of R2RML: [6] hints at an R2RML formalization, but only introduces one rule example without describing the semantics; [7, 8, 9] formalized simpler versions of R2RML as part of SPARQL-to-SQL translations without focusing on R2RML semantics; [10] formalized a reduced version of R2RML as part of mapping patterns optimization without explaining how R2RML semantically operates. These works are discussed in Section 2. This lack of formal semantics hindered further research on R2RML; for example, the correctness of the optimizations that operate on the full R2RML language cannot be proven, only on some parts of the language [10]. Last, performing precise comparisons between R2RML and its alternatives and extensions and identifying their differences is challenging.

Contributions In this paper, we present a declarative, rule-based **formalization of R2RML** that combines both syntactic and semantic aspects. Our goal is to provide precise, unambiguous, and declarative definitions and notations for all the concepts and structures of R2RML through a **Datalog program** [11]. Our Datalog program generates RDF graphs through reasoning, given any 'out-of-the-box' Datalog reasoner, which enables the derivation of implicit knowledge that may not be directly present in the data. Having Datalog rules allows for immediate execution. This approach can, in theory, enable optimizations of rule execution (relying on long-standing Datalog research), reduce redundancy, and facilitate the extension of rules with additional features (e.g., access control, provenance tracking, probabilities, etc.).

We provide a **prototype implementation** that shows the correctness of our semantics in generating RDF graphs. We demonstrate the reasonable efficiency of our Datalog translation by providing **complexity results** and conduct **validation experiments** to assess the correctness of our R2RML Datalog-based semantics, by successfully executing the W3C R2RML test cases [12] using our prototype.

Our work serves as a declarative counterpart to the procedural algorithmic approach provided in the W3C Recommendation Specification for R2RML [4], while our rule-based formal semantics can be extended to formalize mapping languages that depend on R2RML, facilitating efficient comparisons among them.

2. Related Work

We introduce the concept of mappings in OBDA systems. Then, we review related works on the formalization of direct and customized RDB-to-RDF mappings, employed in the materialization and virtualization of RDF graphs [3].

OBDA. In [1], mappings in an OBDA system adapt the mapping techniques in a *Data Integration System* (DIS) [13]. According to [13], a DIS is a system that contains a global schema, a source schema, and mappings that map the source schema to the global schema, as well as in the opposite direction. In [1], the authors concluded that an OBDA system is a subset of a DIS, where the ontology provides the global schema, the data source provides the source schema, and the mappings can only map in the direction of the source schema to the global schema, the so-called *GAV* mappings (Global-As-View).

Following this, there are two methods for answering queries over RDF graphs resulting from applying the mappings to the raw data and the ontology [1]. One is the *Bottom-Up* method, where the RDF graph is fully materialized, and then queries to the ontology are answered on the materialized RDF graph. Such an approach can be time-consuming, prone to redundancy, and require re-materializing the RDF graph if the raw data is modified. The second is called the *Top-Down* method, where the RDF graph is kept virtual. Upon querying the ontology, the query is translated to an SQL query with the guidance of the virtual RDF graph. The SQL query is then delegated to the data source and executed over the data. These two methods became the basis for the materialization and virtualization of RDF graphs [3].

Ontop. Ontop [14, 15, 16] is an OBDA system that adapts the concept of a virtualized RDF graph as a high-level description of RDBs to the user. Despite being able to materialize and virtualize RDF graphs, the Ontop system focuses on virtualizing RDF graphs and using them to answer queries on top of the ontologies. Ontop supports two mapping languages to provide their GAV mappings, Ontop mappings, and R2RML. Ontop mappings are expressed as input/output rules: the input consists of SQL queries over an RDB, and the output is user-defined RDF triples aligned with an ontology. The Ontop mapping language has a concrete theoretical formalization [15], where Datalog and Description logic were employed to define its semantics; similar to this work for R2RML.

R2RML. Sequeda [6] presented initial Datalog rules describing the ontology of R2RML and its logical mapping phase and highlighted the similarity relationship between the core of R2RML and direct mapping with views allowed as input. The authors claimed in [6] that R2RML semantics can be expressed through a fixed number of Datalog rules. However, no R2RML semantics are concretely formalized. The Datalog rules in [6] only demonstrate the different combinations of the R2RML syntax; they neither describe how R2RML operates nor are they executable. It is impossible to study the Datalog rules claimed in the paper as they are not available. After contacting the authors, it was verified that the work was discontinued, and the link for the paper is no longer available.

R2RML and GAV Mappings. Iglesias et al. [10] adapt optimization techniques for materializing RDF graphs from various data sources as output to DIS, where the mappings are described using R2RML or RML. In [10], the mappings are in the form of GAV rules described using *Horn clauses* with functions [11]. The authors relate these GAV rules to R2RML by showing which R2RML concepts belong to which GAV rule. However, these rules do not describe how R2RML operates internally, nor can they be used to execute R2RML. Moreover, these rules miss out on parts of R2RML as graph maps, datatypes, and term types.

R2RML Specification Algorithm. An algorithmic procedural description of R2RML is provided in Section 11 in the R2RML specification [4]. This algorithm details the process of generating an RDF graph from an R2RML triples map. Being heavily nested, this algorithm gives the feeling of falling down a rabbit hole of clicking through notions and subroutines. Initially, the algorithm generates RDF terms with the wrong term types. These types are only corrected in the final steps of the algorithm, which causes confusion. Such ambiguities in the algorithm led to inconsistent behavior of identical R2RML functions across different R2RML implementations.

SPARQL to SQL with R2RML. In the setting of top-down query answering, several approaches formalized SPARQL to SQL rewriting while considering the mappings [7, 9]. The authors in both approaches introduce and formalize a simpler "normalized" version of R2RML that assumes the following: (i) shortcuts for constants and SQL are expanded, (ii) class definitions are replaced by predicate object maps with type as predicate and the class as an object, (iii) predicate object maps with many predicate/object maps are expanded into predicate object maps with one predicate map and one object map, (iv) all referencing object maps have been replaced by a new triple map equivalent to the predicate object map and joins are rather done in the effective SQL query of the logical table, and (v) all triple maps with multiple predicate object maps are replaced by a set of equivalent triple maps with one predicate object map.

Formalizing this normalized R2RML cannot be considered a complete formalization of R2RML. The normalization ignores essential parts of R2RML, such as referencing object maps, graph maps, datatypes, and language tags, and only focuses on parts of R2RML satisfying the goal of using R2RML in the form of GAV mappings. It is ambiguous with this normalization what happens in case an object has a language tag or datatype, or a subject map/predicate-object map has a graph map, or a referencing object map has more join conditions.

Kontchakov et al. [7] formally present SPARQL to SQL rewriting with R2RML being a source of the virtual RDF graph. Normalized R2RML mappings are formalized as GAV rules in Datalog syntax. Rodriguez-Muro and Rezk [9] offer an SQL translation of the generated triples through the normalized R2RML. Similar comments can be made on how that work compares to ours. In a nutshell, they focus on SPARQL-to-SQL rewriting and formalize R2RML as GAV mappings, whereas we focus on a comprehensive formalization of the real-world R2RML through an efficient and executable rule-based approach. The formalization in [9] does not accurately capture the operational behavior of the R2RML language and overlooks several components excluded in its normalized version. For instance, the formalization rules may yield RDF graphs that violate R2RML compliance due to the omission of

IRI-safety conditions on template values.

Similar to [9], [7] employs the normalized R2RML to create virtual RDF graphs. A translation of the triples generation in the normalized R2RML is provided in [7] to SQL, but no concrete general formalization of R2RML is presented.

Priyatna et al. [8] presented Morph: a tool for generating Virtual RDF graphs over RDBs. Morph supports answering SPARQL queries over virtual RDF graphs, by adapting a rewriting for SPARQL-to-SQL queries with user-defined R2RML mappings. Their algorithm shows how SPARQL queries are translated to SQL, considering arbitrary R2RML mappings. R2RML mappings are provided as functions with triples as output. These triples are used as inputs to other functions that generate the SQL query. R2RML is described as part of the SPARQL-to-SQL rewriting, but no formalization of R2RML is provided.

RML. RDF Mapping Language (*RML*) is a declarative mapping language that extends R2RML by supporting various data sources as CSV and JSON [17]. RML is defined as a superset of R2RML with additional features. Min Oo and Hartig proposed a language-agnostic algebra for capturing mapping definitions [18]. They also introduced an algorithm to translate RML mappings into this algebra. However, the translation relies on a simplified, normalized version of RML, similar to the mentioned normalization of R2RML, and overlooks the IRI-safety conditions on template values, resulting in an incomplete formalization of the mapping language. Furthermore, the translation currently does not support RDBs, making it inapplicable for R2RML in its current state. Finally, although the approach proposes optimization strategies for mapping plans, its practical feasibility and performance evaluation remain unaddressed.

3. Preliminaries

This section provides brief overviews of Datalog and R2RML.

Datalog A Datalog program Δ [11, 19] is a set of (possibly recursive) rules (Horn clauses) of the form:

$$\forall \vec{x}.q(\vec{u}) \leftarrow p_1(\vec{t_1}), \ldots, p_m(\vec{t_m}).$$

where $p_1(\vec{t_1}), \ldots, p_m(\vec{t_m})$ and $q(\vec{u})$ are atoms with variables in $\vec{x}$, and relation names $p_1 \ldots p_m$ and q. $\vec{t}$ is a list of *terms* $t_1, \ldots, t_n$ such that each t_i is either a variable or a constant. Also, each variable in $q(\vec{u})$ occurs in some atom $p_i(\vec{t_i})$ which guarantees *safety*. An atom $p(\vec{t})$ is called a *fact* if every $t \in \vec{t}$ is a constant. It is customary to omit the universal quantifiers for brevity. Every Datalog rule has a *Head* $q(\vec{u})$ and a *Body* $p_1(\vec{t_1}), \ldots, p_m(\vec{t_m})$. An atom $p(\vec{t})$ is called an *intensional database atom* (*IDB*) if it occurs in the head of some rule $r \in \Delta$. Otherwise, $p(\vec{t})$ is called an *extensional database atom* (*EDB*). Δ is applied to a database D, where D is a set of EDB facts initially assumed to be true. The output $\alpha(\Delta, D)$ is the IDB facts derived from applying the rules in Δ on D. Databases typically conform to a given *schema* which specifies the available EDB relation names and their arities.

The semantics of a Datalog program Δ is evaluated over a database D, which initially contains a set of EDB facts assumed true. A *ground substitution* replaces the variables in a rule with constants. A *match* occurs when a ground substitution is applied on the body of a rule in Δ, and the resulting facts are added to D. The rule is considered satisfied in Δ if all its matches are added to D. The *least model* $\alpha(\Delta, D)$ is the smallest set of facts containing D that satisfies all the rules in Δ. We compute it by repeatedly applying the *immediate consequence operator*, which adds to the current set of facts all the heads of rules whose bodies matched facts already derived, until no more facts can be added.

R2RML The main component of R2RML mappings is **Triples Maps** that define how to generate RDF triples: (Subject, Predicate, Object). The R2RML vocabulary, expressed through pre-defined IRIs with the prefix **rr**, formally defines each triples map. A triples map has one **Logical Table**, one **Subject Map**, and zero or more **Predicate-Object Maps**. The subject map defines how to generate unique identifiers (*URIs*) used as the subject of all RDF triples generated from this triples map. A predicate-object map has one or more **Predicate Map**(s), which define the rule that generates the URIs for the predicates. Additionally, in every predicate-object map, a predicate map has one or more **Object Map** or/and one or more **Referencing-Object Map**. A referencing object map has one **Parent Triples Map**, which is a triples map. A parent triples map has zero or more **Join Condition**, where each join condition has exactly one **child** value which is a column in the table of the triples map containing the referencing triples map, and exactly one **parent** value which is the column from the logical table of the parent triples map. The object map and referencing object map define how each triple's object is generated.

The subject, predicate, and object maps are **Term Maps** that have one of the **Term Types**: (**IRI**, **Blank Node**, **Literal**). A term map is either **constant-valued** that generates a constant, **column-valued** that is the data value of a column in a logical table, or **template-valued** that is a valid string template having one or more columns of a logical table. Listing 1 shows a simple example of an **R2RML mapping document** with R2RML mappings.

```
<TriplesMap1> a rr:TriplesMap;
rr:logicalTable [ rr:tableName "Student" ];
rr:subjectMap [ rr:template "http://example.com/{Name}" ];
rr:predicateObjectMap [
    rr:predicateMap [rr:constant foaf:name];
    rr:objectMap    [ rr:column "Name" ] ].
```

Listing 1: An R2RML Mapping Document Example.

4. Formal Definition of R2RML Syntax

R2RML [4] defines customized mappings for an RDB $\mathcal{D}$ over a schema $\mathcal{S}$ to an RDF graph $RDF_T \cup RDF_Q$, where RDF_T is a set of triples, each of the form (s, p, o). RDF_Q is a set of triples with named graphs (*quadruples*), each of the form (s, p, o, g). In RDF_T and RDF_Q, s is the subject, p the predicate, o the object, and g the named graph. An important restriction is that $(s, p, o, g) \in (I \cup B) \times I \times (I \cup B \cup L) \times I$, where I, B and L are *RDF terms* and stand for IRIs, blank nodes, and literals respectively. The semantics of RDF are defined in [20]. In this section, we formalize the syntax of R2RML.

Logical Table An R2RML mapping refers to logical tables to retrieve data over the input database schema $\mathcal{S}$. A logical table $\mathcal{LT}$ is either a *Base Table or View Name* in $\mathcal{S}$, or a *R2RML View* that defines a table by executing an SQL query on the input database. $\mathcal{LT}$ always corresponds to an effective SQL query producing the contents of the logical table. In the case of an R2RML view, this is clear by definition; in the case of a base table or view name, the query is a simple $\textbf{SELECT ALL}$ query.

Term Map A term map $Term_{map}$ over a logical table $\mathcal{LT}$ is of the form $(\mathcal{VT}, type, datatype, lantag)$ such that:

- $\mathcal{VT}$ is a value term that is either a constant con, a column name col, or a template $temp$ that defines how to generate an RDF term. Here, $temp$ is an alternating sequence of strings and column names, having the form $(s,\ col_1,\ s_1,\ col_2,\ s_2,\ \ldots,\ col_m,\ s_m)$ for $m \geq 1$.
- $type$ is either **IRI**, **Blank**, **Literal**, or $\perp$, the term type of the RDF term.
- $datatype$ is a valid IRI representing the datatype, or undefined ($\perp$).
- $lantag$ is either a defined valid language tag or undefined $\perp$.
- if $type \neq$ **Literal**, then $datatype$ and $lantag$ must be $\perp$.
- if $type =$ **Literal**, then $datatype = \perp$ iff $lantag \neq \perp$.

The notions of subject term map ST_{map}, predicate term map PT_{map}, object term map OT_{map}, and graph term map GT_{map}, are term maps with additional constraints on the value of *type*. For ST_{map} we have $type = \textbf{IRI}$, $\textbf{Blank}$, or $\perp$. For PT_{map} or GT_{map} we have $type = \textbf{IRI}$ or $\perp$. For OT_{map}, there are no restrictions.

Graph Map A *graph map* G_{map} is a possibly-empty finite set of graph term maps GT_{map}.

Subject Map A *subject map* S_{map} over $\mathcal{LT}$ has the form (ST_{map}, CL, G_{map}) where ST_{map} and G_{map} are as above, and CL is a finite possibly-empty set of IRIs, which will be used as class names.

Predicate-Object Map An *object map* O_{map} is either an object term map OT_{map} as above, or a *referencing* object map, to be defined soon below. Now a predicate-object map PO_{map} over $\mathcal{LT}$ is a triple $(\mathcal{PM}, \mathcal{OM}, G_{map})$ where G_{map} is as above; $\mathcal{OM}$ is a non-empty finite set of O_{map} (object maps); and $\mathcal{PM}$ is a non-empty finite set of PT_{map} (predicate term maps, see above).

Referencing Object Map A *referencing object map* Ref_{Omap} over $\mathcal{LT}$ is a triple[1] $(S_{map_*}, \mathcal{LT}_*, \mathcal{JC})$ such that:

- S_{map_*} is a subject map.
- $\mathcal{LT}_*$ is a logical source.
- $\mathcal{JC}$ is a possible-empty finite set of join conditions, where each $jc \in \mathcal{JC}$ is a pair of valid column names $(child, parent)$ in $\mathcal{LT}$ and $\mathcal{LT}_*$ respectively.
- if $\mathcal{JC} = \emptyset$ then $\mathcal{LT}_* = \mathcal{LT}$.

R2RML Mappings R2RML mappings are defined in a so-called *R2RML Mapping Document* $\mathcal{M}$. Such a document is interestingly an RDF graph in turtle syntax [21]. $\mathcal{M}$ is formed from a finite non-empty set of triples maps. A triples map specifies rules for translating each row of a logical table to zero or more RDF triples or quadruples. Formally, a triples map TM has the form $(\mathcal{LT}, S_{map}, \mathcal{POM})$. Here, $\mathcal{LT}$ is a logical table to be mapped; S_{map} is a subject map over $\mathcal{LT}$, and $\mathcal{POM}$ is a finite set of predicate-object maps PO_{map} over $\mathcal{LT}$.

We provide a summary of the R2RML syntax and the abbreviations used for each component in Appendix 8. Next, we formalize the R2RML semantics.

5. Semantics of Evaluating R2RML Mappings

Let $\mathcal{M}$ be an R2RML mapping document designed for an RDB $\mathcal{D}$ that has a schema $\mathcal{S}$. In R2RML engines, an *R2RML processor* generates an RDF graph $RDF_{\mathcal{M},\mathcal{D}}$ given $\mathcal{M}$ and $\mathcal{D}$. This RDF graph can be either materialized or virtualized as part of an OBDA system. In our work, we aim to use Datalog reasoners as R2RML processors. For each triples map $\mathcal{TM}$ in $\mathcal{M}$ with a logical table $\mathcal{LT}$, we construct a Datalog program $\Delta_{\mathcal{TM}}$ and a database $D_{\mathcal{LT}}$ in what follows.

5.1. Table Facts

Abiteboul et al. [11] introduced a relational model for viewing RDBs in logic programming, by demonstrating two approaches to express RDBs: the *Named* and the *Unnamed* approaches. SQL uses the named approach, whereas Datalog uses the unnamed approach, so we define how we convert from named to unnamed to avoid misunderstanding.

[1]In reality, we reference a parent triples map $\mathcal{PTM}$, but to avoid a circular definition we only introduce what is relevant in $\mathcal{PTM}$, i.e., the logical table and subject map.

Let $A_1, \ldots, A_k$ be the k attributes (column names) of $\mathcal{LT}$, where we agree on a fixed order of the attributes. For every tuple t of table $\mathcal{LT}$ in $\mathcal{D}$, we have an EDB fact $lt(a_1, \ldots, a_k)$ in $\mathcal{D}_{\mathcal{LT}}$, where $a_i = t(A_i)$. Here, lt is the unique name give for $\mathcal{LT}$, used as an EDB atom name of arity k.

Example 1. *Let $\mathcal{LT}$ be the logical table for $\mathcal{TM}$ described in Listing 1.*

```
| Name  |
---------
| Alice |
| Bob   |
```

In this case, $D_{\mathcal{LT}} = \{Student(Alice), Student(Bob)\}$, which are two EDB facts of the two rows in $\mathcal{LT}$.

5.2. Terms Evaluation Rules

Let $Term_{map} = (\mathcal{VT}, type, datatype, lantag)$ be a term map in $\mathcal{TM}$. We provide a set of rules $\Delta^{Term_{map}} \subset \Delta_{\mathcal{TM}}$ with auxiliary IDB atoms required in $\Delta_{\mathcal{TM}}$ to evaluate the value term $\mathcal{VT}$. Below, $lt(\vec{x})$ stands for the EDB atom $lt(x_1, \ldots, x_k)$ with k distinct variables. The rules now are the following.

- if $\mathcal{VT}$ is a constant con:

$$eval_{\mathcal{VT}}(\vec{x}, con) \leftarrow lt(\vec{x}).$$

- if $\mathcal{VT}$ is a column name col, then let j be the index of col in the order O_r of column values of $\vec{x}$ in $lt(\vec{x})$:

$$eval_{\mathcal{VT}}(\vec{x}, x_j) \leftarrow lt(\vec{x}), x_j \neq null.$$

- if $\mathcal{VT}$ is a template $temp$ of the form $(s,\ col_1,\ s_1,\ col_2,\ s_2,\ \ldots,\ col_n,\ s_n)$ for $n \geq 1$:

$$eval_{\mathcal{VT}}(\vec{x}, concat(s, y_1, s_1, y_2, s_2, \ldots, y_n, s_n)) \leftarrow lt(\vec{x}),\ y1, y2 \ldots y_n \neq null.$$

  Here, for each $k \in \{1, \ldots, n\}$, variable y_k equals to x_j in $\vec{x}$, where j is the index of col_k in the order O_r of column values of $\vec{x}$ in $lt(\vec{x})$. Also, $concat$ is a string concatenation function.

These rules construct different value terms from the term maps, depending on the type of $\mathcal{VT}$. $eval_{\mathcal{VT}}$ is a unique predicate name of the atom that evaluates $\mathcal{VT}$. We assume IRI-safe versions for the column values if $type = \textbf{IRI}$. In cases where this condition cannot be guaranteed, each selected column in the head of the rules can instead be wrapped in a function *IRISafe*, which ensures an IRI-safe version of the column value according to Section 7.3 of the R2RML specification[2].

Using functions and built-in predicates in Datalog rules extends beyond pure Datalog. However, such extensions are well-known in the literature [22] and are supported by many modern Datalog engines [23] to enhance expressivity and usability. In our construction, functions are applied only to variables in the heads of rules, and built-in predicates are only applied to variables that are bound in positive body atoms, ensuring rule safety [11].

Example 2. *Let $\mathcal{TM}$ be the triples map in Listing 1, we start the construction of the Datalog program $\Delta_{\mathcal{TM}}$ with the rules evaluating each value term in $\mathcal{TM}$ as follows:*

$$eval_{T}(x0, concat(\text{“http://example.com/”}, x0)) \leftarrow Student(x0).$$
$$eval_{\text{“foaf:name”}}(x0, \text{“foaf:name”}) \leftarrow Student(x0).$$
$$eval_{Name}(x0, x0) \leftarrow Student(x0).$$

where T is the template “http://example.com/{Name}”.

[2]https://www.w3.org/TR/r2rml/#from-template

Next, we further add rules to $\Delta^{Term_{map}}$ for generating the RDF term for a term map $Term_{map}$ having a value term $\mathcal{VT}$ based on the term type *type*. We use the auxiliary IDB atoms, defined in the earlier value-term rules, as input to these rules.

- if $type = \textbf{IRI}$:
$$evalTerm_{IRI}(\vec{x}, concat(\text{“<”}, y, \text{“>”})) \leftarrow eval_{\mathcal{VT}}(\vec{x}, y).$$
- if $type = \textbf{Blank}$:
$$evalTerm_{Blank}(\vec{x}, concat(\text{“\_:”}, y)) \leftarrow eval_{\mathcal{VT}}(\vec{x}, y).$$
- if $type = \textbf{Literal}$:
$$evalTerm_{Literal}(\vec{x}, concat_{datatype,Literal}(\text{“}\,''\,\text{”}, y, \text{“}\,''\,\text{”})) \leftarrow eval_{\mathcal{VT}}(\vec{x}, y).$$

Besides the normal concatenation function, $concat_{datatype,lantag}$ is a function that concatenates the term map with its associated datatype or language tag in case they exist, following the RDF term generation rules in the R2RML specification[3]. $evalTerm_{Type}$ is a unique predicate name of the atom evaluating the RDF term for $Term_{map}$.

Example 3. *We embed $\Delta_{\mathcal{TM}}$ from Ex. 2 with the RDF term generation rules according to their term type:*

$$evalSubject_{IRI}(x0, concat(\text{“<”}, y, \text{“>”})) \leftarrow eval_{T}(x0, y).$$
$$evalPredicate_{IRI}(x0, concat(\text{“<”}, y, \text{“>”})) \leftarrow eval_{\text{“foaf:name”}}(x0, y).$$
$$evalObject_{Literal}(x0, concat_{datatype,Literal}(\text{“}''\text{”}, y, \text{“}''\text{”})) \leftarrow eval_{Name}(x0, x0).$$

5.3. RDF Graph Generation Rules

With the RDF terms generated, we now construct the set of rules $\Delta^{RDF} \subset \Delta_{\mathcal{TM}}$ that generates the RDF Graph.

Let $\mathcal{TM} = (\mathcal{LT}, S_{map}, \mathcal{POM})$ be a triples map with a subject map $S_{map} = (ST_{map}, CL, G_{map})$, and for each predicate object map $PO_{map} \in POM$, $PO_{map} = (\mathcal{PM}, \mathcal{OM}, G_{map*})$. Also, let $G_{map}(\mathcal{TM}) = G_{map} \cup G_{map*}$ be the set of all graph maps in S_{map} and every PO_{map}.

Before defining the rules in Δ^{RDF}, we define the atoms notations *Subject*, *Predicate*, *Object*, and *Graph* using the auxiliary atoms in the defined RDF term generating rules as follows.

Subject Let $ST_{map} \in S_{map}$ be a subject term map with a term type *type* (Section 4), the value of the notation $[\textbf{Subject}(S_{map})]$ is an atom with a name tailored to ST_{map}, determined according to the term type *type*:

- $[\textbf{Subject}(S_{map})] = evalST_{map,IRI}(\vec{x}, s)$ if $type = \textbf{IRI}$ or $\perp$
- $[\textbf{Subject}(S_{map})] = evalST_{map,Blank}(\vec{x}, s)$ if $type = \textbf{Blank}$

Predicate For every predicate term map $PT_{map} \in \mathcal{PM}$, we introduce the notation $\textbf{Predicate}$ representing an atom with a name tailored to PT_{map} and $PO_{map})$ such that:

$$[\textbf{Predicate}(PT_{map}, PO_{map})] = evalPT_{map,IRI}(\vec{x}, p)$$

Object For every object map $O_{map} \in \mathcal{OM}$, if O_{map} is an object term map OT_{map}, we determine the notation $\textbf{Object}$ according to *type* of OT_{map} as follows:

- $[\textbf{Object}(O_{map}, PO_{map})] = evalOT_{map,IRI}(\vec{x}, o)$ if $type = \textbf{IRI}$

[3]https://www.w3.org/TR/r2rml/#generated-rdf-term

- $[\mathbf{Object}(O_{map}, PO_{map})] = eval\,OT_{map,Blank}(\vec{x}, o)$ if *type* $=$ **Blank**

- $[\mathbf{Object}(O_{map}, PO_{map})] = eval\,OT_{map,Literal}(\vec{x}, o)$ if *type* $=$ **Literal** or *type* $=\bot$

Otherwise, if O_{map} is a referencing object map $Ref_{Omap} = (S_{map_*}, \mathcal{LT}_*, \mathcal{JC})$, then **Object** is defined according to $\mathcal{JC}$ such that:

- if $\mathcal{JC} = \emptyset$, then $[\mathbf{Object}(O_{map}, PO_{map})] = [\mathbf{Subject}(S_{map_*})]$
- if $\mathcal{JC} \neq \emptyset$, then for every join condition $jc \in \mathcal{JC}$ with $jc = (col_c, col_p)$ where col_c is a column name in $\mathcal{LT}$ and col_p is a column name in $\mathcal{LT}_*$, we define $\mathbf{JoinCond}(jc)$ as a conjunction of atoms:

  $$[\mathbf{JoinCond}(jc)] = eval_{col_c}(\vec{x}, z), eval_{col_p}(\vec{y}, z)$$

  where z is a fresh variable introduced in both atoms to ensure that the column values in col_c and col_p are the same. To obtain the values of the columns col_c, col_p, we need to ensure the presence of the following two rules in $\Delta_{\mathcal{TM}}$:

  $$eval_{col_c}(\vec{x}, x_i) \leftarrow lt(\vec{x}).$$
  $$eval_{col_p}(\vec{y}, y_j) \leftarrow lt_j(\vec{y}).$$

  Now we define the notation for the set $\mathcal{JC}$ of all join conditions. Assuming that m is the number of join conditions in $\mathcal{JC}$, we define the notation $\mathbf{JoinCond}(\mathcal{JC})$ as a conjunction of every join condition in $\mathcal{JC}$ such that:

  $$[\mathbf{JoinCond}(\mathcal{JC})] = \bigwedge_{i=1}^{i=m} \mathbf{JoinCond}(jc_i)$$

  Following this, we define the notation **Object** as a conjunction of atoms as follows:

  $$[\mathbf{Object}(O_{map}, PO_{map})] = \mathbf{Subject}(S_{map_*}), \mathbf{JoinCond}(\mathcal{JC})$$

Graph For every graph term map $GT_{map} \in G_{map_{\mathcal{TM}}}$:

- if GT_{map} is defined within the subject map S_{map}:

  $$[\mathbf{Graph}(GT_{map}, S_{map})] = eval\,GT_{map,IRI}(\vec{x}, o)$$

- if GT_{map} is defined in the predicate object map PO_{map}:

  $$[\mathbf{Graph}(GT_{map}, PO_{map})] = eval\,GT^{IRI}_{map}(\vec{x}, g)$$

With these notations defined, we introduce the rules in Δ^{RDF} as follows.

Class Rules Let $\Delta^{class} \subset \Delta^{RDF}$ be a set of rules that build the RDF graph in case of the existence of class IRIs in S_{map}. For each class IRI $c \in CL$, we encode it as an EDB atom named $class$, and build the RDF graph accordingly as follows:

- if $G_{map} = \emptyset$:

  $$Triple(s, \text{‘}\texttt{rdf:type}\text{’}, c) \leftarrow [\mathbf{Subject}(S_{map})], class(c).$$

- if $G_{map} \neq \emptyset$, then for each $GT_{map} \in G_{map}$ (Section 4):

  $$Quadruple(s, \text{‘}\texttt{rdf:type}\text{’}, c, g) \leftarrow [\mathbf{Subject}(S_{map})], class(c), [\mathbf{Graph}(GT_{map}, S_{map})].$$

General RDF Rules Let PT_{map} and O_{map} be a predicate term map and an object map belonging to a predicate object map PO_{map}. We define a set of rules $\Delta^N \subset \Delta^{RDF}$ as follows:

- if $G_{map_{\mathcal{TM}}} = \emptyset$:

  $Triple(s, p, o) \leftarrow [\mathbf{Subject}(ST_{map})], [\mathbf{Predicate}(PT_{map}, PO_{map})], [\mathbf{Object}(PT_{map}, PO_{map})].$

- if $G_{map_{\mathcal{TM}}} \neq \emptyset$, then for each $GT_{map} \in G_{map_{\mathcal{TM}}}$:
  - if GT_{map} belongs to the subject map:

$$Quadruple(s, p, o, g) \leftarrow [\mathbf{Subject}(ST_{map})], [\mathbf{Predicate}(PT_{map}, PO_{map})], \\ [\mathbf{Object}(PT_{map}, PO_{map})], [\mathbf{Graph}(GT_{map}, S_{map})].$$

  - if GT_{map} belongs to the predicate object map:

$$Quadruple(s, p, o, g) \leftarrow [\mathbf{Subject}(ST_{map})], [\mathbf{Predicate}(PT_{map}, PO_{map})], \\ [\mathbf{Object}(PT_{map}, PO_{map})], [\mathbf{Graph}(GT_{map}, PO_{map})].$$

These rules generate triples and quadruples by associating each subject with the corresponding predicates, objects, and named graphs as specified in the triples map.

Example 4. *To finalize the construction of $\Delta_{\mathcal{TM}}$ from Ex. 3, we embed $\Delta_{\mathcal{TM}}$ with only one rule from RDF generation rules as follows:*

$$Triple(s, p, o) \leftarrow evalSubject_{IRI}(x0, s), evalPredicate_{IRI}(x0, p), evalObject_{IRI}(x0, o).$$

5.4. RDF Evaluation

To obtain $RDF_{\mathcal{M},\mathcal{D}}$ for $\mathcal{M}$ and $\mathcal{D}$, we construct a Datalog program $\Delta_{\mathcal{M}}$ and a database $D_{\mathcal{M}}$ such that:

- $\Delta_{\mathcal{M}} = \bigcup_{\mathcal{TM} \in \mathcal{M}} \Delta_{\mathcal{TM}}$, where $\Delta_{\mathcal{TM}}$ denotes the Datalog program of the triples map $\mathcal{TM}$.
- $D_{\mathcal{M}} = \bigcup_{\mathcal{TM} \in \mathcal{M}} D_{\mathcal{LT}}$, where $\mathcal{LT}$ is the logical table specified by $\mathcal{TM}$, and $D_{\mathcal{LT}}$ is the corresponding set of EDB facts extracted from $\mathcal{D}$.

The least model $\alpha(\Delta_{\mathcal{M}}, D_{\mathcal{M}})$ (Section 3) contains the RDF graph $RDF_{\mathcal{M},\mathcal{D}}$ as a set of IDB facts of the form $RDF_Q \cup RDF_T$, where RDF_Q and RDF_T are subsets of $\alpha(\Delta_{\mathcal{M}}, D_{\mathcal{M}})$ consisting of IDB facts with predicate names $Quadruple$ and $Triple$, respectively.

Example 5. *The least model $\alpha(\Delta_{\mathcal{M}}, D_{\mathcal{M}})$ of $\Delta_{\mathcal{M}}$ (Ex.4) and $D_{\mathcal{M}}$ (Ex. 1) has the RDF graph $RDF_{\mathcal{M},\mathcal{D}}$ in the form of the facts:*

$Triple($*<ex:Alice>*, *<foaf:name>*, *"Alice"*$)$, $Triple($*<ex:Bob>*, *<foaf:name>*, *"Bob"*$)$

5.5. Translation Complexity

We provide the size complexity of the translated Datalog program for R2RML mappings following our approach.

Let $\mathcal{TM}$ be a triples map with S_{map} and $\mathcal{POM}$. We compute the number of Datalog rules needed in the Datalog program $\Delta_{\mathcal{TM}}$ to compute the RDF graph of $\mathcal{TM}$ as follows:

- For RDF graph generation, 1 rule is required for class RDF generation, and n_{po} rules are needed for general RDF generation, where $n_{po} = \max(n_p, n_o)$, where n_p and n_o denote the total number of predicate maps and object maps in $\mathcal{POM}$, respectively.
- For S_{map}, 2 rules are required to generate the RDF term.

- For all predicate maps in $\mathcal{POM}$, $2n_p$ rules are needed to generate the RDF terms, where n_p is the number of predicate maps.
- For the object maps in $\mathcal{POM}$, the number of Datalog rules required is $\sum_{i=1}^{n_o}(2 + jc_i)$, where n_o denotes the number of object maps. Each object map requires two rules to generate the RDF term. If the object map is a referencing object map, each join condition in that object map requires two additional rules to be evaluated, where jc_i is the total number of join conditions for the ith object map.
- Let n_g be the number of graph maps in $S_{map} \cup \mathcal{POM}$. The total number of rules needed to generate the RDF term from these graph maps is $2n_g$.

As a conclusion, the total number of rules $R_{\mathcal{TM}}$ for $\mathcal{TM}$ can be computed through the following equation:

$$R_{\mathcal{TM}} = 3 + n_{po} + 2(n_p + n_o + n_g) + 2\sum_{i=1}^{n_o} jc_i$$

We observe that the size of $\Delta_{\mathcal{TM}}$ is at most polynomial in the size of $\mathcal{TM}$. As a conclusion, the size of $\Delta_{\mathcal{M}}$ is polynomial in the size of $\mathcal{M}$.

6. R2RML Datalog Implementation

To validate the correctness and efficiency of our R2RML Datalog-based semantics, we developed a prototype that utilizes the mappings and data parsers of *RMLMapper* [24]: a Java tool that generates RDF graphs for RML [17] and also supports R2RML. With RMLMapper's parsers, our prototype can translate any input RDB and R2RML mappings into a Datalog program ready for execution.

R2RML to Datalog translation. Our prototype processes any R2RML mapping document $\mathcal{M}$ and translates it to a corresponding Datalog program. In addition, the prototype translates the specified RDB tables in $\mathcal{M}$ into corresponding facts files. Once generated, the Datalog program and facts files can be passed to any 'out-of-the-box' Datalog reasoner, which generates the corresponding RDF graph of $\mathcal{M}$ through reasoning. Our prototype is available in the Github repository (https://github.com/dtai-kg/R2RML2Datalog-Translator).

Datalog execution. We rely on Soufflé as a Datalog reasoner due to its efficiency, scalability, and broad support of user-defined functions [25, 23], which are required in our semantics. Soufflé compiles Datalog programs into C++ code, which enables parallel execution. Soufflé supports user-defined functions in the form of C++ code wrapped in C functions, however, Soufflé does not support RDF concepts. We implemented the required user-defined functions to ensure the proper execution of our Datalog program in Soufflé. The functions are available in the GitHub repository of our prototype.

Validation. To evaluate the correctness of our R2RML Datalog-based semantics in generating RDF graphs, we executed all R2RML test cases [12] on our prototype with Soufflé as a Datalog reasoner, and MySQL as the RDBS to create the logical tables for all test cases. From the 62 official R2RML test cases (TCs), 14 TCs assess the correctness of how the tool parses the input R2RML mapping document and data. However, since our prototype is built on RMLMapper's parser, these 14 TCs are not relevant as they are covered by RMLMapper; we refer to the R2RML implementation report in [26]. Our prototype successfully generated the correct RDF graph in all of the 48 TCs used. Our GitHub repository (https://github.com/dtai-kg/R2RML2Datalog-Tests) contains a folder for each of the 48 test cases (TCs). Each folder includes: the R2RML mapping document, the corresponding translated Datalog program in Soufflé syntax, the RDB in MySQL syntax, its corresponding facts files as input for Soufflé, the expected RDF output, and the actual output produced by Soufflé reasoning.

7. Conclusions and Future Work

In this work, we provided the complete formal syntax formalization of R2RML. We presented a translation of R2RML mappings to Datalog programs that captures the semantics of R2RML, and discussed its complexity. In addition, we developed a prototype implementation that translates an RDB and a set of R2RML mappings into EDB facts and a Datalog program. These EDB facts and Datalog program generate the intended RDF graph through reasoning with any 'out-of-the-box' Datalog reasoner that supports user-defined functions. We validated the correctness of our R2RML semantics by executing all official R2RML test cases using our prototype implementation.

Our work paves the way for conducting theoretical research on R2RML and studying its properties. By expressing R2RML mappings in terms of Datalog, we enable their integration with a broad range of Datalog-based reasoning capabilities, e.g., efficient query answering, access control, and provenance tracking.

In the future, we plan to conduct efficiency experiments comparing our R2RML Datalog-based approach with prominent R2RML tools. We also plan to provide a single "universal" Datalog program which, unlike the approach in this paper, does not require a translation step to convert the R2RML mappings into a custom Datalog program. Instead, a single, fixed Datalog program is used, while the R2RML mappings and the RDB are encoded as input EDB facts to this program. However, this approach comes at the cost of increased reasoning complexity, as it requires support for both negation and recursion. Lastly, we plan on extending our Datalog-based semantics to support the RML language by adapting the current semantics to support additional input data formats.

Acknowledgments

Dimou and Elhalawati's contributions to this research were partially supported by Flanders Make, the strategic research centre for the manufacturing industry. All three authors are partially supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-4 and Grammarly for Grammar and spelling checks. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking Data to Ontologies, in: S. Spaccapietra (Ed.), Journal on Data Semantics X, volume 4900 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2008, pp. 133–173. doi:10.1007/978-3-540-77688-8_5.

[2] M. Arenas, A. Bertails, E. Prud'hommeaux, J. F. Sequeda, A Direct Mapping of Relational Data to RDF, W3C Recommendation, 2012. URL: https://www.w3.org/TR/rdb-direct-mapping/.

[3] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review, Journal of Web Semantics 75 (2023). doi:10.1016/j.websem.2022.100753.

[4] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, Working Group Recommendation, 2012. URL: http://www.w3.org/TR/r2rml/.

[5] J. F. Sequeda, M. Arenas, D. P. Miranker, On directly mapping relational databases to RDF and OWL, in: Proceedings of the 21st International Conference on World Wide Web (WWW '12), ACM, 2012, p. 649–658. doi:10.1145/2187836.2187924.

[6] J. F. Sequeda, On the semantics of R2RML and its relationship with the direct mapping, in: Proceedings of the 12th International Semantic Web Conference Posters & Demonstrations Track (ISWC-PD '13), volume 1035, CEUR-WS.org, 2013, p. 193–196. URL: https://ceur-ws.org/Vol-1035/iswc2013_poster_4.pdf.

[7] R. Kontchakov, M. Rezk, M. Rodríguez-Muro, G. Xiao, M. Zakharyaschev, Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime, in: The 13th International Semantic Web Conference (ISWC 2014), Springer, 2014, p. 552–567. doi:10.1007/978-3-319-11964-9_35.

[8] F. Priyatna, O. Corcho, J. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: Proceedings of the 23rd international conference on Worldwide Web (WWW '14), ACM, Seoul, Korea, 2014, pp. 479–490. doi:10.1145/2566486.2567981.

[9] M. Rodriguez-Muro, M. Rezk, Efficient SPARQL-to-SQL with R2RML mappings, Journal of Web Semantics 33 (2015) 141–169.

[10] E. Iglesias, S. Jozashoori, M.-E. Vidal, Scaling up knowledge graph creation to large and heterogeneous data sources, Journal of Web Semantics 75 (2023). doi:10.1016/j.websem.2022.100755.

[11] S. Abiteboul, R. Hull, V. Vianu (Eds.), Foundations of Databases: The Logical Level, 1st ed., Addison-Wesley Longman Publishing Co., Inc., 1995. URL: http://webdam.inria.fr/Alice/.

[12] M. Hausenblas, B. Villazón-Terrazas, R2RML and Direct Mapping Test Cases, W3C Note, W3C, 2012. URL: https://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/.

[13] M. Lenzerini, Data integration: A theoretical perspective, in: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, 2002, pp. 233–246.

[14] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL Queries over Relational Databases, Semantic Web Journal 8 (2017) 471–487. doi:10.3233/SW-160217.

[15] M. Rodríguez-Muro, R. Kontchakov, M. Zakharyaschev, Ontology-Based Data Access: Ontop of Databases, in: The 12th International Semantic Web Conference (ISWC 2013), Springer, 2013, pp. 558–573. doi:10.1007/978-3-642-41335-3_35.

[16] G. Xiao, D. Lanti, R. Kontchakov, S. Komla-Ebri, E. Güzel-Kalaycı, L. Ding, J. Corman, B. Cogrel, D. Calvanese, E. Botoeva, The Virtual Knowledge Graph System Ontop, in: The 19th International Semantic Web Conference (ISWC 2020), Springer, 2020, pp. 259–277. doi:10.1007/978-3-030-62466-8_17.

[17] 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: Proceedings of the 7th Workshop on Linked Data on the Web, volume 1184, CEUR-WS.org, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[18] S. M. Oo, O. Hartig, An Algebraic Foundation for Knowledge Graph Construction (Extended Version), CoRR abs/2503.10385 (2025). doi:10.48550/ARXIV.2503.10385.

[19] G. Gottlob, G. Orsi, A. Pieris, M. Simkus, Datalog and Its Extensions for Semantic Web Databases, in: Reasoning Web. Semantic Technologies for Advanced Query Answering: 8th International Summer School 2012 Proceedings, Springer, 2012, pp. 54–77. doi:10.1007/978-3-642-33158-9_2.

[20] C. Gutierrez, C. Hurtado, A. O. Mendelzon, Foundations of semantic web databases, in: Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '04), ACM, 2004, p. 95–106. doi:10.1145/1055558.1055573.

[21] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, G. Carothers, RDF 1.1 Turtle – Terse RDF Triple Language, Recommendation, World Wide Web Consortium (W3C), 2014. URL: http://www.w3.org/TR/turtle/.

[22] S. Ceri, G. Gottlob, L. Tanca, What You Always Wanted to Know About Datalog (And Never Dared to Ask), IEEE Transactions on Knowledge and Data Engineering 1 (1989) 146–166. doi:10.1109/69.43410.

[23] B. Ketsman, P. Koutris, Modern Datalog Engines, Foundations and Trends® in Databases 12 (2022) 1–68. doi:10.1561/1900000073.

[24] A. Dimou, T. De Nies, R. Verborgh, E. Mannens, R. Van de Walle, Automated Metadata Generation for Linked Data Generation and Publishing Workflows, in: Proceedings of the 9th Workshop on Linked Data on the Web (LDOW@WWW), volume 1593, 2016. URL: https://ceur-ws.org/Vol-1593/article-04.pdf.

[25] Z. Fan, J. Zhu, Z. Zhang, A. Albarghouthi, P. Koutris, J. M. Patel, Scaling-up in-memory datalog processing: observations and techniques, Proceedings of the VLDB Endowment 12 (2019) 695–708. doi:10.14778/3311880.3311886.

[26] D. Chaves-Fraga, P. Heyvaert, R2RML Implementation Report, 2012. URL: https://kg-construct.github.io/r2rml-implementation-report/, accessed: 2025-05-27.

8. Appendix

R2RML Syntax Summary The following table provides a bottom-up summary of the components of the R2RML syntax, along with their corresponding abbreviations and structure:

R2RML Component	**Abbreviation**	**Structure**
logical table	$\mathcal{LT}$	Base Table or View Name
Term map	$Term_{map}$	$(\mathcal{VT}, type, datatype, lantag)$
Value term	$\mathcal{VT}$	con, col , or a $temp$
Constant value	con	constant
Column name	col	column name in the schema
Template	$temp$	$(s, col_1, s_1, col_2, s_2, \ldots, col_n, s_n)$
Term type	$type$	**IRI**, **Blank**, **Literal** or $\perp$
Datatype	$datatype$	a datatype IRI, or $\perp$
Language tag	$lantag$	a valid language tag, or $\perp$
Subject term map	ST_{map}	same as a term map
Predicate term map	PT_{map}	same as a term map
Object term map	OT_{map}	same as a term map
Graph term map	GT_{map}	same as a term map
Graph map	G_{map}	a set of graph term maps
Subject map	S_{map}	(ST_{map}, CL, G_{map})
Class IRIs	CL	zero or more class IRI
Predicate map	$\mathcal{PM}$	a set of predicate term maps
Object map	O_{map}	OT_{map} or a referencing object map
Object map set	$\mathcal{OM}$	one or more object map
Predicate object map	PO_{map}	$(\mathcal{PM}, \mathcal{OM}, G_{map})$
Predicate object map set	$\mathcal{POM}$	one or more PO_{map}
Child	$child$	a valid column name in $\mathcal{LT}$
Parent	$parent$	a valid column name in $\mathcal{LT}_*$
join condition	jc	$(child, parent)$
join condition set	$\mathcal{JC}$	zero or more jc
Referencing object map	Ref_{Omap}	$(S_{map_*}, \mathcal{LT}_*, \mathcal{JC})$
Triples map	$\mathcal{TM}$	$(\mathcal{LT}, S_{map}, \mathcal{POM})$
R2RML mapping document	$\mathcal{M}$	one or more $\mathcal{TM}$