Conformance Test Cases for the RDF Mapping Language (RML)

Pieter Heyvaert[1], David Chaves-Fraga[2][0000−0003−3236−2789], Freddy Priyatna[2][0000−0002−9292−4010], Oscar Corcho[2][0000−0002−9260−0753], Erik Mannens[1], and Anastasia Dimou[1]

[1] IDLab, Department of Electronics and Information Systems, Ghent University – imec

[2] Ontology Engineering Group, Universidad Politécnica de Madrid

Abstract. Knowledge graphs are often generated using rules that apply semantic annotations to certain data sources. Software tools execute these rules and generate or virtualise the corresponding RDF-based knowledge graph. R2RML is a W3C recommended language to write such rules for relational databases, while RML provides an extension of R2RML to incorporate support for other data sources, such as data in CSV, XML, and JSON format. When developing a tool to execute rules, for each set of rules the expected knowledge graph should be generated, i.e., the tool conforms to the language's specification. As part of its standardisation process, a set of test cases was created for R2RML to assess tools conformance. However, no test cases were proposed so far for knowledge graphs generated from other data sources. In this work, we generated an initial set of reusable test cases to assess knowledge graphs generation from heterogeneous data sources. These test cases rely on R2RML test cases and can be used by any tool, regardless of the programming language, to evaluate the tool's conformance to the RML specification. We tested the conformance of two RML processors: the RMLMapper and CARML. The results show that the RMLMapper passes all test cases regarding CSV, XML, and JSON format, and most test cases for RDBs, but fails the ones for automatic datatyping of literals. CARML passes most test cases regarding the CSV, XML, and JSON format, with exception of test cases that deal, for example, with multiple RDF terms generation. Developers can now determine how conformant their tools are to the RML specification and users can use this conformance results to determine the most suitable tool for their use cases.

Keywords: RML · R2RML · Test Cases

1 Introduction

Knowledge graphs are often generated based on rules that apply semantic annotations to certain data. For example, the DBpedia knowledge graph is generated by applying classes and predicates of the DBpedia ontology to Wikipedia [1].

Software tools execute these rules and generate corresponding RDF triples and quads [2], which materialize knowledge graphs. In the past, custom scripts prevailed, but lately rule-driven tools emerged. Such tools distinguish the rules that define how RDF terms and triples are generated from the tool that executes them. R2RML [3] is the W3C recommended language to define such rules for generating knowledge graphs from data in relational databases (RDBs). An R2RML processor is a system that, given a set of R2RML rules and a relational database, generates an output RDF dataset. Examples of R2RML processors are, e.g., Ultrawrap [4], Morph-RDB [5], Ontop [6], and XSPARQL [7]. A subset of them was included in the RDB2RDF Implementation Report [8] to determine their conformance to the R2RML specification [3], i.e., the correct knowledge graph is generated for a set of rules and certain relational database.

Extensions and adaptations were applied to R2RML to account for other types of data sources, given that R2RML is focused on relational databases only, such as RML [9], XSPARQL [7], xR2RML [10], KR2RML [11], and D2RML [12]. RML provides an extension of R2RML to support heterogeneous data sources, including different formats, e.g., CSV, XML, JSON, and access interfaces, e.g., files and Web APIs. Similarly, RML processors emerged that execute RML rules, such as the RMLMapper[4], CARML[5], GeoTriples[6], and Ontario[7]. Unlike R2RML, there are no test cases available to determine the conformance of the processors to the RML specification. As a result, the processors are either not tested or only tested with custom test cases, which do not necessarily assess every aspect of the specification. Consequently, no implementation report is available that allows comparing the different processors that generate knowledge graphs from heterogeneous data sources based on the conformance to the specification. This way it is hard to determine the most suitable processor for a certain use case.

In this work, (i) we focused on RML and introduce an initial set of RML test cases, which contains 297 test cases based on the existing R2RML test cases. However, instead of only considering relational databases as data sources, as it occurs for the R2RML test cases, we also consider data in CSV, XML, and JSON format. Furthermore, (ii) we tested the conformance of the RMLMapper and CARML: every test case is executed by each processor and we noted if the generated knowledge graph matches the expected one. The corresponding implementation report is available at `http://rml.io/implementation-report`. This allows to determine which processor is the most suitable for a certain use case. For example, do users want a processor that supports the complete specification, or do they prefer a processor that does not support certain aspects of the specification, but executes the rules faster?

The test cases results shows that the RMLMapper passes all test cases regarding CSV, XML, and JSON format, and most test cases for RDBs, but fails

[3] Some of those available in the report are no longer actively maintained and used
[4] RMLMapper, `https://github.com/RMLio/rmlmapper-java`
[5] CARML, `https://github.com/carml/carml`
[6] GeoTriples, `https://github.com/LinkedEOData/GeoTriples`
[7] Ontario, `https://github.com/WDAqua/Ontario`

the test cases for automatic datatyping of literals. CARML passes most test cases regarding CSV, XML, and JSON format, except of the test cases that deal, for example, with multiple RDF terms generation. Users can now determine how conformant the different processors are to the RML specification and use this conformance to determine the most suitable processor for their use cases.

The remainder of the paper is structured as follows. In Section 2, we discuss related work. In Section 3, we discuss the test cases. In Section 4, we elaborate on the test cases execution and results. In Section 5, we conclude the paper.

2 Related work

In this section, we describe the related work that is relevant to the paper. First, we explain the most important knowledge graph generation language specifications, including R2RML and RML, and processors that execute those rules. Second, we discuss the differences between R2RML and RML. Finally, we describe the R2RML test cases, how they are defined and implemented and their corresponding implementation report with results of a few processors.

2.1 Knowledge graph generation languages and tools

R2RML [3] is the W3C recommended language for describing rules to generate RDF from data in RDBs. Currently, many tools support this specification. These tools follow either an Extract-Transform-Load (ETL) process, where a knowledge graph is materialized, e.g., DB2Triples[8] and R2RMLParser[9], or they provide virtual RDF views, focusing more on formalizing the translation from SPARQL to SQL and optimizing the resulted SQL query, e.g., Morph-RDB[10] and Ontop[11].

We describe in more details pioneering tools for executing R2RML rules: DB2Triples is a tool for extracting data from relational databases, semantically annotating the data extracts according to R2RML rules and generating Linked Data. The R2RMLParser [13] deals in principle with incremental Linked Data generation. Each time a knowledge graph is generated, not all data is used, but only the one that changed (so-called incremental transformation). Morph-RDB [5] and Ontop [6] adapt the algorithm defined by Chebotko, Lu, and Fotouhi [14] on SPARQL-to-SQL translation, using the information provided by the R2RML rules. Both apply several semantic optimizations (e.g., self join elimination) that generate efficient SQL queries to speed up the evaluation time.

RML [9] is defined as an extension of R2RML to specify rules for generating knowledge graphs from data in different formats, such as CSV, JSON, XML, and different access interfaces, e.g., open data connectivity and Web APIs [15]. Different other languages build upon RML for generating knowledge graphs from heterogeneous data sources, e.g., xR2RML [10] or RMLC [16].

[8] DB2triples, `https://github.com/antidot/db2triples`
[9] R2RMLParser, `https://github.com/nkons/r2rml-parser`
[10] Morph-RDB, `https://github.com/oeg-upm/morph-rdb`
[11] Ontop, `https://github.com/ontop/ontop`

A set of processors that support the RML specification are proposed. The RMLMapper is a Java library and command line interface that executes RML rules to generate RDF. Following the same approach, CARML executes RML rules, but also includes its own extensions, such as MultiTermMap (to deal with arrays) and XML namespace (to improve XPath expressions). GeoTriples is a processor that generates and executes RML rules for generating RDF from geospatial data from different sources. The processor supports data stored in raw files (shapefiles, CSV, KML, GML, and so on), but also geospatial RDBs such as PostGIS[12] and MonetDB[13]. The generated RDF is based on well-known geospatial vocabularies, such as GeoSPARQL [17] and stSPARQL [18]. Ontario [19] is a federated query processor that uses RML rules to transform heterogeneous data sources during the query processing. Basically, the processor performs the generation using RML during the query processing step and executes federated SPARQL queries over the resulted RDF graphs. These processors are evaluated using ad-hoc examples or feasibility approaches, but a thorough representation of their capabilities is not provided. For that reason, we notice that RML test cases are needed to assess the capabilities of the different processors.

2.2 R2RML and RML differences

Table 1. The differences between R2RML and RML

	R2RML	**RML**
input reference	Logical Table	Logical Source
data source language	SQL (implicit)	Reference Formulation (explicit)
value reference	column	Logical reference (valid expression acc. Reference Formulation)
iteration	per row (implicit)	per record (explicit – valid expression acc. Reference Formulation)

RML is an extension of R2RML and, thus, follows the core concepts of R2RML's specification, such as Triples Maps, Term Maps, Subject Maps, and so on. However, there is a difference on the reference to the data to support heterogeneous data sources with respect to their format, e.g. CSV, XML, JSON, and access interface, e,g. files or Web APIs (see Table 1).

Logical Source A Logical Source extends R2RML's Logical Table and describes the input data source used to generate the RDF. The Logical Table is only able

[12] `https://postgis.net/`
[13] `https://www.monetdb.org/`

to describe relational databases, whereas the Logical Source defines different heterogeneous data sources, including relational databases.

Reference Formulation As RML is designed to support heterogeneous data sources, data sources in different formats needs to be supported. One refers to data in a specific format according to the grammar of a certain formulation, which might be path and query languages or custom grammars. For example, one can refer to data in an XML file via XPath and in a relational database via SQL. To this end, the *Reference Formulation* was introduced indicating the formulation used to refer to data in a certain data source.

Iterator In R2RML it is specified that processors iterate over each row to generate RDF. However, as RML is designed to support heterogeneous data sources, the iteration pattern cannot always be implicitly assumed. For example, iterating over a specific set of objects is done by selecting them via a JSONPath expression. To this end, the *Iterator* was introduced which determines the iteration pattern over the data source and specifies the extract of data used to generate RDF during each iteration. The iterator is not required to be specified if there is no need to iterate over the input data.

Logical Reference When referring to values in a table or view of a relational database, R2RML relies on column names. However, as RML is designed to support heterogeneous data sources, rules may also refer to elements and objects, such as in the case of XML and JSON. Consequently, references to values should be valid with respect to the used reference formulation. For example, a reference to an attribute of a JSON object should be a valid JSONPath expression. To this end, (i) the `rml:reference` is introduced to replace `rr:column`, (ii) when a template is used, via `rr:template`, the values between the curly brackets should have an expression that is valid with respect to the used reference formulation, and (iii) `rr:parent` and `rr:child` of a Join Condition should also have an expression that is valid with respect to the used reference formulation.

2.3 W3C recommendations and their test cases

In the context of Semantic Web, several specifications were recommended by W3C, such as SPARQL [20], RDF [2], SHACL [21], Direct Mapping of relational data to RDF (DM) [22], and R2RML [3]. Each of these specifications has several related tools that support them. A set of test cases was defined for each one of them (SPARQL test cases[14], RDF 1.1 test cases[15], SHACL test cases[16], and R2RML and Direct Mapping test cases[17], respectively) that provides useful information to choose the tool that fits better to certain needs. It is also a relevant

[14] `https://www.w3.org/2001/sw/DataAccess/tests/r2`
[15] `http://www.w3.org/TR/rdf11-testcases/`
[16] `http://w3c.github.io/data-shapes/data-shapes-test-suite/`
[17] `https://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/`

step in the standardisation process of an technology or specification. We describe the R2RML in more details as it is related to the scope of this paper.

Determining the conformance of tools executing R2RML rules in the process of RDF generation is a step to provide objective information about the features of each tool. For this reason, the R2RML test cases [23] were proposed. It provides a set of 63 test cases. Each test case is identified by a set of features, such as the SQL statements to load the database, title, purpose, specification reference, review status, expected result, and corresponding R2RML rules. All the test cases are semantically described using the RDB2RDF-test[18] and Test Metadata Vocabulary[19]. Several R2RML processors were assessed for their conformance with the R2RML specification running the test-cases. The results are available in the R2RML implementation-report [8]. The results are also annotated semantically using the Evaluation and Report Language (EARL) 1.0 Schema[20].

3 RML Test cases

In this section, we propose test cases to determine the conformance of RML processors to the RML specification. The proposed test cases are based on the R2RML test cases, but they take into account different heterogeneous data sources and the corresponding differences in RML (see Section 2.2). Our preliminary set of test cases includes (i) adjusted R2RML test cases for relational databases (including MySQL[21], PostgreSQL[22], and SQL Server[23]) and (ii) new test cases for files in the CSV, XML (with XPath as the reference formulation), and JSON format (with JSONPath as the reference formulation). The test cases are described at `http://rml.io/test-cases/` and the corresponding files are available at `https://github.com/rmlio/rml-test-cases`. In Section 3.1, we describe the data model that is used to represent the test cases. In Section 3.2, we elaborate on the different files making up a test case. In Section 3.3, we discuss the differences between the R2RML and RML test cases.

3.1 Data model

We describe the test cases semantically to increase their reusability and sharability. To this end, we created a semantic data model[24], with as main entity the test case (see Figure 1). For each test case, the following details are described: unique identifier, title, description, relevant aspect of the RML specification, data sources (optional), expected knowledge graph or error, and RML rules.

[18] `http://purl.org/NET/rdb2rdf-test#`
[19] `https://www.w3.org/TR/2005/NOTE-test-metadata-20050914/`
[20] `https://www.w3.org/TR/EARL10/`
[21] `https://www.mysql.com/`
[22] `https://www.postgresql.org/`
[23] `https://www.microsoft.com/en-us/sql-server/`
[24] `http://rml.io/test-cases/#datamodel`

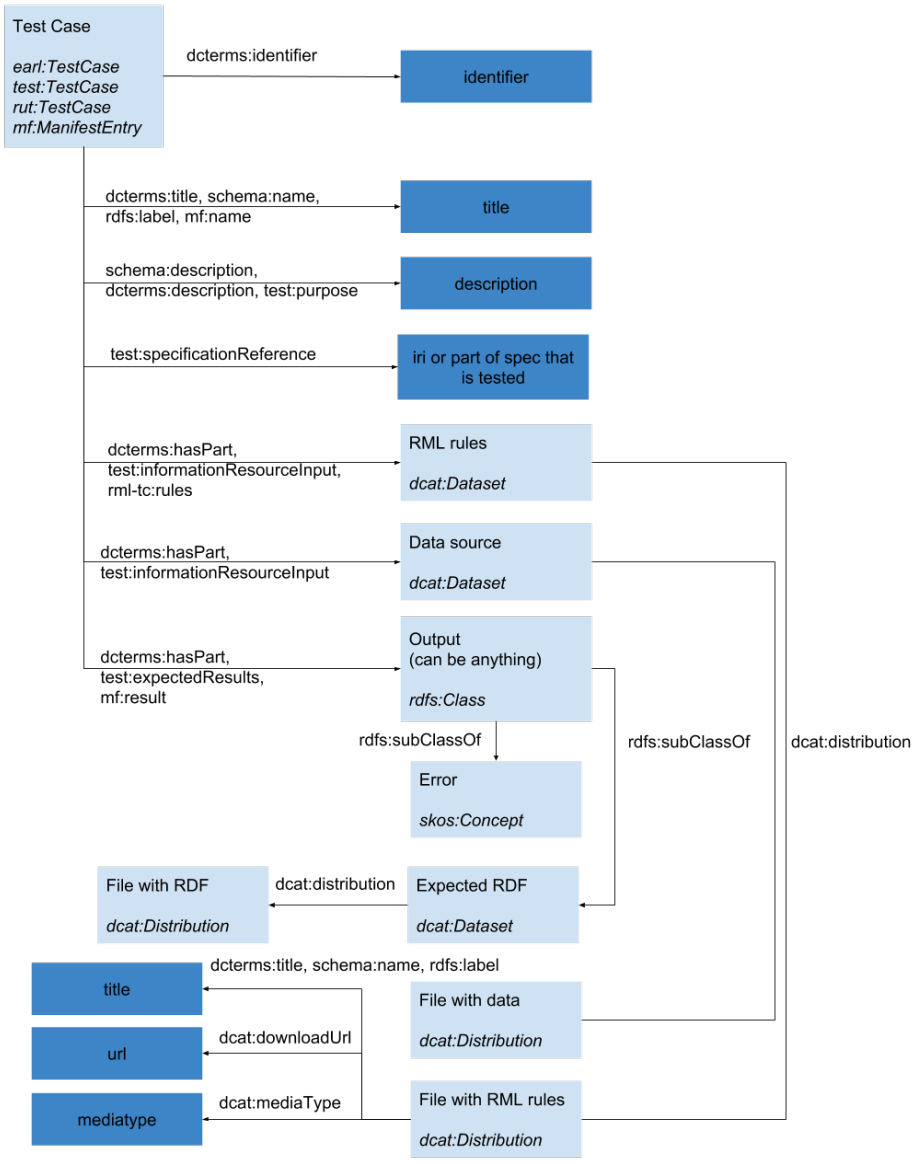

Fig. 1. Data model of the RML test cases

To provide the corresponding semantic descriptions, the model uses mostly the Evaluation and Report Language (EARL) 1.0 Schema[25], the Test case mani-

[25] `https://www.w3.org/TR/EARL10/`, with prefix `earl`

fest vocabulary[26], the Test Metadata vocabulary[27], and the Data Catalog Vocabulary[28]. A test case is annotated with the classes `earl:TestCase`, `test:TestCase`, and `mf:ManifestEntry`. The identifier, title, description, and the specific aspect of the RML specification that is being tested are added as datatype properties. The files that are provided as input to the tools are linked to the test cases via `test:informationResourceInput` and `dcterms:hasPart`. The file with the RML rules is also linked via `rml-tc:rules`[29]. The objects of these properties are of the class `dcat:Dataset`, which in turn link to a `dcat:Distribution` that includes a link to a file. The expected output, whether that is a knowledge graph or an error, is linked via `test:expectedResults`, `mf:result`, and `dcterms:hasPart`. In the case of a knowledge graph, the object of these properties is a `dcat:Dataset`, linked to a `dcat:Distribution`, to describe the file containing the graph. In the case of an error, we link to the expected error.

3.2 Test case files

Each test case consists of a set of files that contain the input data sources, the RML rules, and the expected RDF output. In practice, the files are organized as follows: all files for a single test case are contained in a single folder.

There are three types of files for each test case:

- 0 or more **data source files** for CSV (with extension .csv), XML (with extension .xml), and JSON (with extension .json), or 1 file with SQL statements to create the necessary tables for relational databases (called `resource.sql`);
- 1 **file with the RML rules** (in Turtle format, called `mapping.ttl`); and
- 0 or 1 **file with the expected RDF** (in N-Quads format, called `output.nq`).

Distinct test cases assess different behaviours of the processors. Certain test cases assess the behaviour of the tools when (i) the required data sources are not available, and others when (ii) an error occurs and no output is generated. In the former, no data sources files or SQL statements are provided. In the latter, no file with the expected RDF is provided. The test cases are independent of how the processors materialize the knowledge graph: a data dump, as done by the RMLMapper, or on the fly, as done by Ontario [24].

3.3 Differences with R2RML test cases

For most R2RML test cases, we created an RML variant for CSV, XML, JSON, MySQL, PostgreSQL, and SQL Server, leading to 6 RML test cases per R2RML test case. For R2RML test cases that focus on specific features of SQL queries, we only created 3 RML test cases, i.e., for MySQL, PostgreSQL, and SQL Server.

[26] `http://www.w3.org/2001/sw/DataAccess/tests/test-manifest#`, with prefix `mf`
[27] `https://www.w3.org/2006/03/test-description#`, with prefix `test`
[28] `https://www.w3.org/TR/vocab-dcat/`, with prefix `dcat`
[29] `http://rml.io/ns/test-cases`, with prefix `rml-tc`

For test cases with CSV, XML, and JSON files as data sources, we created the corresponding files with the data based on the tables of the relational databases. For CSV, we used the table created by the SQL statements of the R2RML test case and stored it as a CSV file. For XML, the name of the table was used for the root of the XML document and every row of the table was used to create an XML element. Within this element, elements were created for each column and their values are the values of the corresponding columns in the table. For JSON, we followed a similar approach as XML. The file contains a JSON object at the root with the name of the table as the only attribute. This attribute has as value an array, where each element of the array corresponds with a row in the table. For each row, attributes were created for each column and their values are the values of the corresponding columns in the table.

Data errors. 2 of the R2RML test cases expect a data error to happen, e.g., when the subject IRI of an entity cannot be generated. In this case, an error is thrown and no knowledge graph is generated. With RML for entities where no subject IRI can be generated there is also no output generated, but, in contrast to R2RML, for the other entities the corresponding output is still generated. Therefore, for the corresponding RML test cases the processors can still throw an error, but the generation of the knowledge graph must not be halted.

Inverse expressions. 3 of the R2RML test cases are designed to test the use of inverse expressions[30]. However, inverse expressions are only used to optimize the knowledge graph generation and no differences are observed in the generated knowledge graph. Thus, whether inverse expressions are used by a processor or not cannot be verified by such test cases. Thus, we do not include them for RML.

SQL-specific features. 18 of the R2RML test cases focus on specific features of SQL queries, e.g., a duplicate column name in a SELECT query. As there are no corresponding RML test cases for CSV files, XML files with XPath, and JSON files with JSONPath, we only provide 54 corresponding test cases for MySQL, PostgreSQL, and SQL Server.

Null values. 1 of the R2RML test cases tests null values in the rows. However, a corresponding RML test case cannot be provided for the CSV and XML format, because both formats do not support null values.

Spaces in columns. 1 of the R2RML test cases is designed to test the behaviour when dealing with spaces in the columns of the SQL tables. However, a corresponding RML test case cannot be provided for the XML format, because it does not allow spaces in names.

In total, we have 297 test cases: 39 for CSV, 38 for XML, 41 for JSON, and 180 for relational databases. Of these 297, 255 test cases expect an knowledge graph to be generated, while 36 expect an error that halts the generation.

[30] `https://www.w3.org/TR/r2rml/#inverse`

4 Test case execution and results

In this section, we describe the executing of the test cases and their results for two RML processors: the RMLMapper and CARML. We detail on (i) the method for running the test cases and obtaining the results, which are annotated semantically, and (ii) the implementation report similar as proposed for R2RML[31]. After this, we present the results of the RMLMapper and CARML.

4.1 Method

We define a method to run the RML test cases over RML processors and generate the corresponding results. The main goal is to facilitate the testing process and provide a general solution for running the test cases over other RML processors that can be developed in the future. The method consists of two main steps: (i) assessing if a processor passes every test case and (ii) annotating the obtained results as RDF using the EARL Schema through a set of YARRRML rules [25].

We implemented a Java-based tool for checking the conformance of the test cases over different RML processors. At this moment, it is able to evaluate the test cases for JSON, CSV and XML formats. We relied on the test framework of each processor to execute the RDB test cases. If a new processor wants to be added to test its conformance, the framework only needs to have access to the corresponding output of each test-case. Then it checks, using the same method for all processors, if the output is the same as the correct one, if an error that is expected has really thrown, etc. The code is available online[32] together with a Web page showing the results of all tested tools[33]. It generates as output two CSV files with the results and the metadata needed to generate the corresponding RDF.

The results are semantically annotated, as the test cases, using the EARL Schema. Each test case evaluated by a tool is annotated as an `earl:Assertion` with the properties: `earl:subject` with the URI of the tool, `earl:assertedBy` with the identifier of who performed the evaluation, `earl:test` with the URI of the test and `earl:result` with the result of the test-case.

Three types of results are possible: "passed", "failed", and "inapplicable". **Passed** (`earl:passed`) is used either when the actual output matches the expected output when no error is expected, or when the tool throws an error when an error is expected. **Failed** (`earl:failed`) is used either when the actual output does not match the expected output if no error is expected, the processor returns an error trying to execute a test or the tool does not throw error if an error is expected. **Inapplicable** (`earl:inapplicable`) is used when the tool clearly states that specific features used in a test case are not supported. The results also provide its type (`earl:TestResult`) and its mode (`earl:automatic` for all these cases). We created a set of YARRRML rules to generate these annotations following the EARL Schema that, using the outputs of the test cases.

[31] `https://www.w3.org/TR/rdb2rdf-implementations/`
[32] `https://github.com/RMLio/rml-implementation-report`
[33] `http://rml.io/implementation-report`

4.2 Results

Table 2. Results of the RMLMapper

RMLMapper	**CSV**	**XML**	**JSON**	**MySQL**	**PostgreSQL**	**SQL Server**	**Total**
passed	39	41	38	55	55	55	283
failed	0	0	0	5	5	5	15
inapplicable	0	0	0	0	0	0	0

Table 3. Results of CARML

CARML	**CSV**	**XML**	**JSON**	**MySQL**	**PostgreSQL**	**SQL Server**	**Total**
passed	29	28	28	0	0	0	85
failed	10	10	13	0	0	0	33
inapplicable	0	0	0	60	60	60	180

We perform the test-cases over two processors: RMLMapper and CARML. In Table 2 we show the results for the RMLMapper processor. It passes all CSV, JSON and XML test cases, but fails in the same 5 test cases for the RDBs. The failures are related to the automatic datatyping of literal for RDBs specified by R2RML[34]. RMLMapper expects to pass the failed test-cases in next versions of the processor. The effort prediction to pass these test-cases is not very much since the failures depend on the general processor, not on the used RDBMS. Once they have been solved for one RDBMS they will automatically pass over the rest.

In Table 3 we show the results for the CARML processor. It partially passes the CSV, JSON and XML test cases, but it does not provide support for any of the RDBs test cases. The failures are related to the unsupported for multiple Subject Maps, multiple Predicate Maps, and Named Graphs. The developers of the tool declare that CARML will support these features in next versions of the processor. However, at the moment of writing, we do not have any information about if CARML will provide support for RDBs.

Finally, we can declare that testing a RML processor with the defined cases and analysing the obtained results offers a general view of the current status of it. These results also give useful information to the tool developers on knowing where they should put their effort to improve the conformance of the processor.

[34] `https://www.w3.org/TR/r2rml/#dfn-natural-rdf-literal`

5 Conclusion

With the introduction of an initial set of RML test cases (i) developers can determine how conformant their RML processors are to the RML specification, and (ii) users can use the test cases results to select the most appropriate processor for a specific use case. The results of the test cases execution with the RMLMapper and CARML show that the CSV, XML, and JSON formats are almost fully supported, but RDBs cause difficulties or are not supported at all. However, the aspects of the RML specification whose test cases failed can be supported by the processors in the future through (minor) implementation updates.

Our set of test cases is based on the R2RML test cases, and therefore, it covers a big part of the RML specification, as it is based on R2RML. However, as the R2RML test cases focus on relational databases, they do not take into account the specifics of hierarchical data formats, such as nested structures in JSON and XML files, which can be used with RML. Therefore, further research should be directed towards creating new test cases that tackle these specifics taking into account the differences between the different hierarchical formats and their corresponding reference formulations.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Research Foundation – Flanders (FWO), and the European Union. The work presented in this paper is partially supported by the Spanish 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 an FPI grant (BES-2017-082511).

References

[1] Jens Lehmann et al. "DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web* 6 (2015), pp. 167–195.

[2] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. Recommendation. World Wide Web Consortium (W3C), Feb. 2014. URL: http://www.w3.org/TR/rdf11-concepts/.

[3] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. http://www.w3.org/TR/r2rml/. W3C, 2012.

[4] Juan F. Sequeda and Daniel P. Miranker. "Ultrawrap: SPARQL execution on relational data". In: *Web Semantics: Science, Services and Agents on the WWW* (2013). ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2013.08.002. URL: http://www.sciencedirect.com/science/article/pii/S1570826813000383.

[5] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. "Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using Morph". In: *23rd International Conference on WWW*. 2014. ISBN: 978-1-4503-2744-2.

[6] Diego Calvanese et al. "Ontop: Answering SPARQL Queries over Relational Databases". In: *Semantic Web Journal* (2017).

[7] Stefan Bischof et al. "Mapping between RDF and XML with XSPARQL". In: *Journal on Data Semantics* (2012). ISSN: 1861-2040. DOI: 10.1007/s13740-012-0008-7. URL: http://dx.doi.org/10.1007/s13740-012-0008-7.

[8] Souripriya Das, Seema Sundara, and Richard Cyganiak. *RDB2RDF Implementation Report*. W3C Note. https://www.w3.org/TR/rdb2rdf-implementations/. W3C, 2012.

[9] Anastasia Dimou et al. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *LDOW*. 2014.

[10] Franck Michel et al. "Translation of Relational and Non-relational Databases into RDF with xR2RML". In: *WEBIST*. 2015.

[11] Jason Slepicka et al. "KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources." In: *COLD*. 2015.

[12] Alexandros Chortaras and Giorgos Stamou. "D2RML: Integrating heterogeneous data and web services into custom RDF graphs". In: *Proceedings of the LDOW. CEUR, ceur-ws. org* 2073 (2018).

[13] Nikolaos Konstantinou et al. "Exposing scholarly information as Linked Open Data: RDFizing DSpace contents". In: *The Electronic Library* 32.6 (2014), pp. 834–851. DOI: 10.1108/EL-12-2012-0156.

[14] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. "Semantics preserving SPARQL-to-SQL translation". In: *Data & Knowledge Engineering* 68.10 (2009), pp. 973–1000.

[15] Anastasia Dimou et al. "Machine-interpretable Dataset and Service Descriptions for Heterogeneous Data Access and Retrieval". In: *Proceedings of the 11th International Conference on Semantic Systems*. SEMANTICS '15. ACM, 2015. ISBN: 978-1-4503-3462-4. DOI: 10.1145/2814864.2814873. URL: http://doi.acm.org/10.1145/2814864.2814873.

[16] David Chaves-Fraga et al. "Virtual Statistics Knowledge Graph Generation from CSV files". In: *Emerging Topics in Semantic Technologies: ISWC 2018 Satellite Events*. Vol. 36. Studies on the Semantic Web. IOS Press, 2018, pp. 235–244.

[17] Robert Battle and Dave Kolas. "Geosparql: enabling a geospatial semantic web". In: *Semantic Web Journal* 3.4 (2011), pp. 355–370.

[18] Manolis Koubarakis and Kostis Kyzirakos. "Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL". In: *Extended Semantic Web Conference*. Springer. 2010, pp. 425–439.

[19] Maria-Esther Vidal et al. "Semantic Data Integration of Big Biomedical Data for Supporting Personalised Medicine". In: *Current Trends in Se-*

mantic Web Technologies: Theory and Practice. Springer, 2019, pp. 25–56.

[20] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. Recommendation. World Wide Web Consortium (W3C), Mar. 2013. URL: https://www.w3.org/TR/sparql11-query/.

[21] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. Recommendation. World Wide Web Consortium (W3C), 2017. URL: https://www.w3.org/TR/shacl/.

[22] Marcelo Arenas et al. *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. https://www.w3.org/TR/rdb-direct-mapping/. W3C, Sept. 2012.

[23] Boris Villazón-Terrazas and Michael Hausenblas. *R2RML and Direct Mapping Test Cases*. W3C Note. http://www.w3.org/TR/rdb2rdf-test-cases/. W3C, 2012.

[24] Anastasia Dimou et al. "What Factors Influence the Design of a Linked Data Generation Algorithm?" In: *Proceedings of the 11th Workshop on Linked Data on the Web*. Ed. by Tim Berners-Lee et al. Apr. 2018. URL: http://events.linkeddata.org/ldow2018/papers/LDOW2018_paper_12.pdf.

[25] Pieter Heyvaert et al. "Declarative Rules for Linked Data Generation at your Fingertips!" In: *Proceedings of the 15th ESWC: Posters and Demos*. 2018.