Rule-driven inconsistency resolution for knowledge graph generation rules

Pieter Heyvaert *, Ben De Meester, Anastasia Dimou and Ruben Verborgh
IDLab, Department of Electronics and Information Systems, Ghent University – imec, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
E-mail: pheyvaer.heyvaert@ugent.be

Abstract. Knowledge graphs contain annotated descriptions of entities and their interrelations, and are often generated based on rules that state how certain data sources are semantically annotated. Inconsistencies are introduced in these graphs when ontology terms are (re)used without adhering to the restrictions defined by the ontologies, affecting the quality of the graphs. Rules and ontologies are two possible root causes for these inconsistencies. Methodologies and tools were proposed to detect and resolve these inconsistencies. However, they either require the complete knowledge graph, which is not always available in a time-constrained situation; or assume that only the rules can be refined and not the ontologies. In the past, we proposed a rule-driven methodology to detect and resolve inconsistencies without requiring the complete knowledge graph, but it only allows applying a predefined set of refinements to the rules. Therefore, we propose with this paper a rule-driven methodology, extending our previous work, that considers refinements for both rules and ontologies. In this work, we provide (i) a detailed description of our methodology and its implementation; and (ii) our findings when applying the methodology to two real-life use cases: DBpedia and DBLP. The use cases show that our methodology provides valuable insights when determining which refinements should be applied to the rules and ontologies, such as the entities that need to most attention when applying refinements, and the specific ontology terms and definitions that are involved in a lot of inconsistencies and that therefore might be problematic.

Keywords: inconsistency, knowledge graph, methodology, resolution, rule-driven

1. Introduction

Knowledge graphs contain annotated descriptions of entities and their interrelations by using ontologies [22]. The graphs can be published as Linked Data [4] using the Resource Description framework (RDF) [7] as data representation. A common way to generate these graphs is by applying semantic annotations to certain data sources, that contain these entities, via a set of rules. The semantic annotations are added via terms defined by ontologies, such as *classes*, *properties*, and *datatypes*. The rules state which data fractions align with which ontology terms, and, thus, determine how the knowledge graph is using existing data sources and ontologies. The syntax and the grammar of the rules are determined by a knowledge graph generation language, such as R2RML [8], RML [11], and SPARQL-Generate [18]. As an example, consider the existing data sources in Tables 1 and 2, providing the id and name of two people and the id, name, and manufacturer of two pieces of furniture. In Listing 2, the definitions of an ontology are listed that describe people and furniture. These definitions are used to annotate the existing data via the rules in Listing 3. The resulting knowledge graph can be found in Listing 1.

Restrictions on the use of the ontology terms can exist, either defined via the ontology term's definitions or via the interpretation of the ontology's axioms as restrictions [2, 17]. For example, axioms can state that certain classes are disjoint, such as `ex:Person` and `ex:Furniture` (see Listing 2, line 3), or there are definitions that put restrictions on the domain and range of properties, such as `ex:Person` which is in the domain of `ex:name` (see Listing 2, line 5). Inconsistencies are introduced in graphs when ontology terms are used without adhering to the restrictions, af-

*Corresponding author. E-mail: pheyvaer.heyvaert@ugent.be.

1570-0844/0-1900/$35.00 © 0 – IOS Press and the authors. All rights reserved

id	name	country
0	Ash	Belgium
1	Misty	France

Table 1
Existing data source with the id, name, and country of people

id	name	manufacturer
21	Chair	Comfy Inc.
22	Sofa	Seated Company

Table 2
Existing data source with the id, name, and manufacturer of furniture

fecting the quality of the graphs. There are three possible root causes identified for these inconsistencies: (i) *data sources*, i.e., the data contains inconsistencies [19]; (ii) *rules*, i.e., they introduce new inconsistencies by, for example, not using the suitable ontology terms [12, 21]; and (iii) *ontology definitions*, i.e., they do not model the domain as desired [21]. In this work, we focus on the latter two root causes.

Previous research efforts introduced methodologies and tools to identify inconsistencies in knowledge graphs [3, 17]. Although this enables resolving the inconsistencies in the graph itself, it does not fix the root issue, so the same inconsistencies reappear when the knowledge graph is regenerated. Therefore, methodologies and tools have been developed to identify inconsistencies in the knowledge graph generation rules [12, 21]. Compared to the solutions that work directly on the knowledge graph, these find the inconsistencies in less time, while simultaneously identifying the rules, ontology terms and definitions causing them.

```
1  ex:0 a ex:Furniture.
2  ex:0 a ex:Person.
3  ex:0 ex:name "Ash".
4  ex:0 ex:madeIn "Belgium".
5  ex:1 a ex:Furniture.
6  ex:1 a ex:Person.
7  ex:1 ex:name "Misty".
8  ex:1 ex:madeIn "France".
9  ex:21 a ex:Furniture.
10 ex:21 a ex:Person.
11 ex:22 a ex:Furniture.
12 ex:22 a ex:Person.
```

Listing 1: Linked Data generated by applying the rules in Listing 3 on the data in Tables 1 and 2

```
1  ex:Furniture is a class
2  ex:Person is a class
3  ex:Person and ex:Furniture are disjoint
4  ex:name is a property
5  ex:Person is the domain of ex:name
6  ex:madeIn is a property
7  ex:Furniture is the domain of ex:madeIn
8  xsd:string is the datatype of ex:madeIn
9  ex:livesIn is a property
10 ex:Person is the domain of ex:livesIn
11 ex:name has values that are uppercase
```

Listing 2: Ontology that describes people and furniture

In our running example, triple 1 states that the first entity is of the class `ex:Furniture` and triple 2 states that the first entity is also of the class `ex:Person` (see Listing 1). However, the ontology has a definition that the classes `ex:Furniture` and `ex:Person` are disjoint (see Listing 2). This inconsistency is caused due to the combination of rules 3 and 4, which generate these triples. Rule 3 annotates an entity with the class `ex:Furniture` and Rule 4 annotates an entity with the class `ex:Person` (see Listing 3 and Table 1). This is not consistent with the ontology, as `ex:Furniture` and `ex:Person` are disjoint, which then leads to the aforementioned inconsistency.

The rules or ontology definitions need to be refined to resolve inconsistencies, but this is not straightforward. On the one hand, which rules should be refined? For example, is it rule 3, rule 4 or both of them? How should they be refined: should the class in rule 3 be `ex:Person` or another class? What are the effects of updating these rules? For instance, would that lead to other inconsistencies, such as caused by rules 3 and 5 if the entity is not longer annotated with the class `ex:Person`? On the other hand, if the ontology can be updated, which definitions should be refined? For example, how should the definitions be refined: remove definition 5 or add another definition to expand the domain of `ex:name`? Even more, the situation aggravates when the set of rules and their relationships grow, or multiple and more complex ontologies are used. For example in the case of DBpedia where there are more than 2,000 inconsistencies, which rules should be inspected first, when considering the more than 600,000 rules, the more than 700 classes and more than 2,800 properties of the DBpedia ontology[1]?

In previous work, we proposed a rule-driven methodology to resolve inconsistencies by automatically re-

[1] http://dbpedia.org/ontology/

```
1  every row represents an entity (Table 1)
2  every entity has as IRI http://example.com/ concatenated with the value in column `id' (Table 1)
3  every entity is of the class ex:Furniture (Table 1)
4  every entity is of the class ex:Person (Table 1)
5  the value of the column `name' is related to the entity via the property ex:name (Table 1)
6  the value of the column `country' is related to the entity via the property ex:madeIn (Table 1)
7  property ex:madeIn has datatype xsd:Integer (Table 1)
8  every row represent an entity (Table 2)
9  every entity has as IRI http://example.com/ concatenated with the value in column `id' (Table 2)
10 every entity is of the class ex:Furniture (Table 2)
11 every entity is of the class ex:Person (Table 2)
```

Listing 3: Rules that define how to generate the Linked Data in Listing 1 based on the data in Table 1

fining the rules [12]. Inconsistencies are detected by analyzing the rules and predefined refinements are automatically applied to resolve the inconsistencies. However, it assumes that used ontologies align with the envisioned semantic model of the user, which is not always the case [21]. Furthermore, when a high number of rules are involved in the inconsistencies users have no insights in which rules should be inspected first, in the case of a manual resolution. This leads to the following research question:

> How can we rank rules and ontology terms for inspection to improve the manual resolution of inconsistencies with a rule-driven methodology?

In this work, we address this question by introducing a rule-driven methodology that extends our previous work [12]. The rules and ontology terms are ranked in the order that they should be inspected by an expert based on a score. The score takes into account the number of inconsistencies a rule or ontology term is involved in. Our proposed methodology consists of the following steps:

1. detect inconsistencies by analyzing the rules, instead of the knowledge graph;
2. rank rules and ontology terms in the order that they should be inspected by experts;
3. refine rules and ontologies based on refinements given by experts;
4. generate the knowledge graph;
5. detect inconsistencies in this graph; and
6. refine rules and ontologies to resolve inconsistencies.

Our novel contributions include in particular:

- a score to rank rules considering the number of inconsistencies and relationship between rules;
- a score to rank ontology terms considering the number of inconsistencies;
- an implementation of the methodology using an existing knowledge graph generation language;
- a detailed analysis of these rankings and possible refinements on different use cases, e.g. DBpedia.

The remainder of this article is structured as follows. In Section 2, we discuss the state of the art. In Section 3, we elaborate on the methodology. In Section 4, we discuss the implementation. In Section 5, we present our findings about two use cases. Finally, in Section 6, we conclude the article.

2. Related work

Knowledge graphs can be generated via rules in different languages (see Section 2.1). These graphs can contain inconsistencies, which can be detected by assessing the graphs' quality (see Section 2.2). Once these inconsistencies are detected they are resolved to improve the knowledge graph (see Section 2.3).

A number of research efforts assume that knowledge graphs are materialized via the Resource Description Framework (RDF) [7, 14]. RDF uses a graph-based model with as core structure a set of triples. Each triple consists of a subject, predicate, and object. A set of triples are called an RDF graph.

2.1. Knowledge graph generation rules

Knowledge graph generation languages, e.g., R2RML [8], RML [11], and SPARQL-Generate [18], offer a

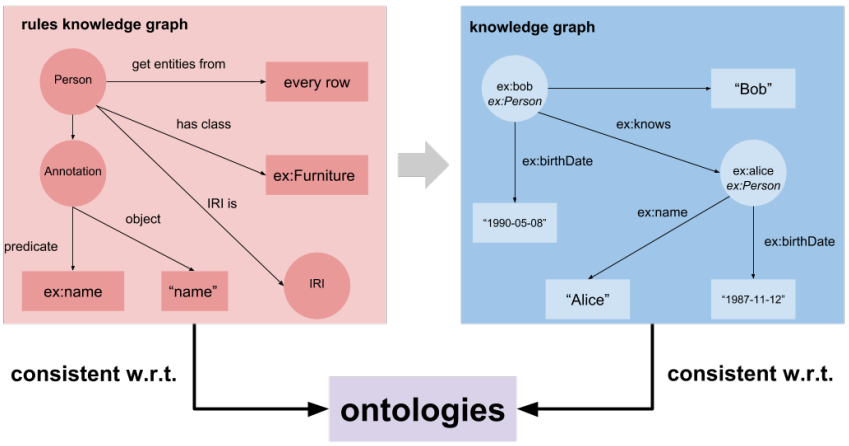

Figure 1. Rules form a knowledge graph that also has to be consistent w.r.t. the use ontologies

declarative way to define how knowledge graphs are generated from raw data. [R2]RML rules are written in RDF, forming themselves a knowledge graph, i.e., the so-called *rules knowledge graph* (see Fig. 1).

Knowledge graphs are often constructed by consistently applying the terms of certain ontologies, i.e., the graphs respect the restrictions imposed by the definitions in the ontologies. This, in turn, is also applicable to rules knowledge graphs, i.e., they need to be constructed so that the knowledge graphs, which are generated by executing these rules, respect the restrictions imposed by the used ontologies.

2.2. Knowledge graph quality assessment

There exists a number of methodologies to assess the quality of knowledge graphs. On the one hand, there are methodologies applied directly to the knowledge graph, based on e.g., crowdsourcing [1], the comparison of the results of queries [15, 17], inference rules [3, 21], or custom characteristics [20]. These methodologies have access to the complete knowledge graph and can identify every inconsistency. However, they require the graph to be available, which is not always possible in a time-constrained situation [13].

On the other hand, there are methodologies applied to the rules that generate knowledge graphs, such as our previous work [12] which is based on the aforementioned comparison of the results of queries originally applied to the knowledge graph [17]. Such methodologies result in faster execution times, but not all inconsistencies can be identified, as some of them depend on the actual data values in the graph.

Kontokostas et al. [17] introduced a list of common patterns, which are called *constraint types* [6], that can be used to detect inconsistencies in a knowledge graph. These constraint types can also be used to find inconsistencies in rules for knowledge graph generation.

As these patterns were designed for the resulting knowledge graph, not all of them can be applied to the rules. More specific, when a pattern refers to specific values, then we can only identify inconsistencies when the rules specify a constant value for these values, i.e., no references to the actual data. For example, if we have the constraint type for the range of the value of a property and the value in the knowledge graph is based on a number in the existing data source, then we cannot know if the number is within the range without inspecting the data source. However, we know if a constant value is within the range, as this constant value does not depend on the data source.

2.3. Inconsistency resolution

We can identify the following distinct approaches for resolving inconsistencies, which assume RDF as the means to represent knowledge graphs: inconsistencies are resolved by updating (i) the knowledge graph directly, i.e., *triple-level*, (ii) the rules that generate the knowledge graph, i.e., *rule-level*, or (iii) the ontology definitions, i.e., *ontology-level*.

2.3.1. Triple-level

Inconsistencies identified in knowledge graphs can be resolved by refining the graphs directly. For RDF, this means adding, removing, or refining certain triples.

Sieve [20] is a framework for quality assessment and fusion of knowledge graphs. The quality assessment task is realized through a flexible module, where the user can choose which characteristics of the data indi-

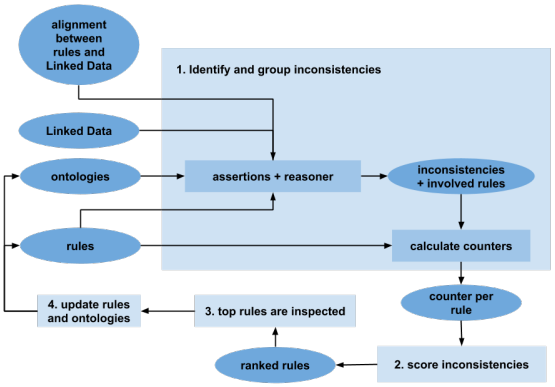

Figure 2. Steps followed during analysis of DBpedia rules and used ontologies [21]

cate higher quality, how this quality is quantified, and how it should be stored in the system. The output of this task is a set of scores used during the fusion task. This helps users in determining which data should be removed or transformed.

CLAMS [15] is a system to discover and resolve inconsistencies in knowledge graphs. It defines an inconsistency as a minimal set of triples that cannot coexist. The system identifies all inconsistencies through the execution of a set of queries. The involved triples are ordered based on the number of inconsistencies they participate in. Removing any triple from that set will resolve the inconsistency. Users use the system's graphical user interface (GUI) to update or remove the triples. The GUI offers the option to see all the inconsistencies that a triple participates in and why it is part of a particular inconsistency. Once a triple is updated or removed the set of inconsistencies and involved triples are updated.

These tools enable resolving inconsistencies in the knowledge graph, but the inconsistencies in the rules remain. Consequently, when regenerating the knowledge graph with the unaltered rules, the same inconsistencies will be present again. Thus, methodologies were investigated that are applied on the rules directly.

2.3.2. *Rule-level*

Inconsistencies identified in knowledge graphs can be resolved by refining the rules, instead of the generated graphs. Knowledge graph refinement through the use of external methods [22], occurs when the source of knowledge to refine the knowledge graph, i.e., ontologies, is not part of the original knowledge graph, i.e., the rules. In our previous work [12], we proposed a uniform, iterative, incremental assessment and refinement methodology for RML rules that produces a high-quality knowledge graph, and more specific, an RDF dataset (see Fig. 3). It is created by applying the assessment process, normally applied to the knowledge graph, to the knowledge graph generation rules. This allows discovering inconsistencies in the knowledge graph, before it is generated, which might take a considerable amount of time [13]. The methodology consists of the following steps:

1. Inconsistencies are detected via the rules, as it would have been done with the actual dataset.
2. Rules are automatically refined (step 2A in Fig. 3) and re-assessed to detect new inconsistencies.
3. The refined version of the rules are used to generate the knowledge graph.
4. The generated knowledge graph is assessed, using the same quality assessment framework, to find remaining inconsistencies.
5. Rules can be refined again to resolve these inconsistencies.

The same constraint types, normally applied to an RDF dataset, are considered. For example, instead of validating the predicate against the triple's subject and object, the rules that define how the subject, predicate, and object are generated and validated. The properties and classes in the rules are identified and their schemas are used to generate test cases, which are similar to the test cases for the actual dataset.

We adjusted the assessment queries by Kontokostas et al. [17] to apply them to the rules. However, some of the constraint types normally applied to a dataset rely on the final values or refer to the complete dataset and, thus, can only be validated after the rules are executed. For example, if the literal value of a certain property is within a given range. We refer to the original work [17] for details about the alignment between the violation patterns and the rules that should be refined and how.

Although we introduced a rule-driven inconsistency resolution methodology, the aforementioned two steps focus on [R2]RML [8, 11], while a similar methodology might also be applied on knowledge graph generation rules defined using a different language. More, it assumes that the used ontologies correctly define the user's envisioned semantic model, which is not always the case [21]. User intervention can be considered to decide if the rules or ontologies need to be refined. However, the methodology does not provide a way to guide users regarding which rules should be inspected first and how these rules and ontologies can be refined.

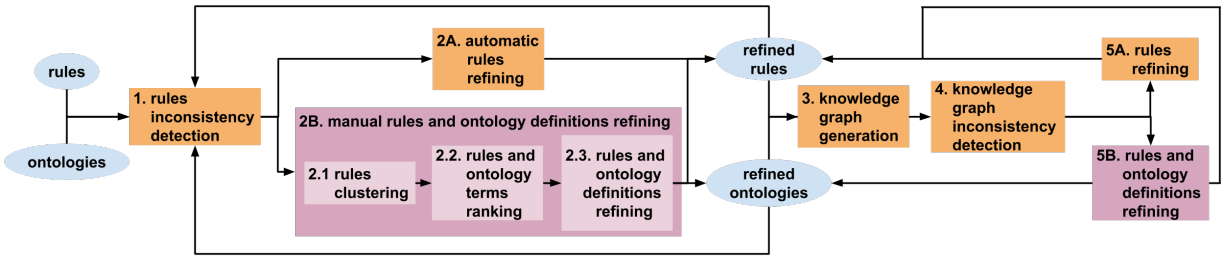

Figure 3. The steps of the rule-driven methodology of our previous and current work

2.3.3. *Ontology-level*

Paulheim [21] performed a *data-driven analysis* on the DBpedia rules and concluded that refinements might not only be needed on rules, but also no the ontology definitions. An overview of the steps followed during his analysis can be found here and in Fig. 2:

1. **Identifying and grouping inconsistencies.** Relation assertions and their subject's and object's types are extracted from the knowledge graph. The rules that contribute to these assertions and types are identified. They are used, together with the ontologies, to determine the inconsistencies through reasoning, i.e., which combination of assertions and types are inconsistent with respect to the ontologies' terms definitions. All rules are marked that contribute to a specific inconsistency to keep two counters for each rule: how often the rule generates an assertion involved in an inconsistency (i_m) and how often it does not (c_m).
2. **Scoring inconsistencies.** A score ($score(m)$) is calculated for all rules as the harmonic mean of the logarithmic support ($logs(m)$) and confidence ($c(m)$). The logarithmic support is calculated as $logs(m) = \frac{log(i_m+1)}{log(N+1)}$ and the confidence as $c(m) = \frac{i_m}{i_m+c_m}$. The final score is calculated as follows $score(m) = \frac{2 \cdot logs(m) \cdot c(m)}{logs(m)+c(m)}$.
3. **Inspect top rules.** The rules are ranked based on the scores and the top rules are manually inspected to determine if the rules or ontology definitions should be refined.
4. **Update rules or ontologies.** The refinements are applied, which results in refined rules and ontology definitions. The process can be repeated to determine and fix still remaining or newly identified inconsistencies using the refined rules and ontology terms definitions.

These steps fill a gap in our previous work, because it provides a set of top ranked rules that should be manually inspected by an expert, similar to Sieve and CLAMS. However, the need for the complete knowledge graph increases the execution time, which makes it not suitable for use cases that are dealing with time-constraints [13]. Furthermore, the alignment between the knowledge graphs and which rules generated them needs to be available.

3. Methodology

We propose a methodology to resolve inconsistencies in knowledge graphs that occur due to the (i) rules that define how the graphs are generated, or (ii) ontology terms that annotate the data. To achieve this, we extend our previous work on assessing rules knowledge graph [12], by ranking the rules, as proposed in the data-driven analysis [21] (see Section 2.3.2), and by ranking the ontology terms.

Our methodology (i) detects inconsistencies by analyzing the rules, instead of the knowledge graph; (ii) clusters the rules involved in an inconsistency; (iii) ranks the rules and ontology terms in the order that they should be inspected by experts; (iv) refines the rules and ontologies based on the refinements given by the experts; (v) generates the knowledge graph; (vi) detects inconsistencies in the knowledge graph; and (vii) refines the rules and ontologies based on these inconsistencies. An overview of the methodology can be found here and in Fig. 3:

1 **Rules inconsistency detection.** We validate the rules. The outcome is a set of inconsistencies and the (combination of) rules and ontology terms involved in these inconsistencies.

2 **Rules and ontology definitions refinement.**

2.1 **Rules clustering.** We cluster the rules per entity, because rules concerning a single entity impact each other, and, thus, their refinements too.

2.2 **Rules and ontology terms ranking.** We calculate a score for each rules cluster and ontology term. We use this score to provide a ranking for each rule and ontology term.
2.3 **Rules and ontology definitions refinement.** We inspect the top rules and ontology definitions that correspond with the terms manually and apply necessary refinements.

3 **Knowledge graph generation.** We use the refined rules and ontology definitions to generate the knowledge graph.
4 **Knowledge graph inconsistency detection.** We validate the knowledge graph. The outcome of this step is a set of inconsistencies and the (combination of) rules and ontology terms involved in these inconsistencies.
5 **Rules and ontology definitions refinement.** We further refine the rules and ontology definitions to resolve the newly discovered inconsistencies.

In the remainder of this section, we provide a detailed explanation of the aforementioned steps.

3.1. Rules inconsistency detection

We validate the rules to determine which inconsistencies are present with respect to used ontologies (see step 1 in Fig. 3). This step is analogous to the first step of our previous work and consists of three substeps:

1 Instantiated constraints are generated by aligning the constraint types with the axioms from the ontologies. This is done assuming that axioms can be interpreted as constraints [17] (see Section 2.2). This substep does not depend on how the constraints are described, other constraints can also be used, such as SHACL constraints [16].
2 Rules that could make an instantiated constraint fail are grouped, based on the types of rules that are involved in each constraint.
3 The groups are analyzed to assess if they respect the related constraint. If this is not the case, an inconsistency is found. For each inconsistency that is present, we call the group of rules that cause it the *involved rules*, and the related ontology terms *involved ontology terms*.

Example Consider the axiom on line 3 in the ontology (see Listing 2). The corresponding instantiated constraint is as follows: for every rule that annotates an entity with the class `ex:Furniture`, there should not exist a rule that annotates that same entity with the class `ex:Person`, and vice versa. We determine all subsets of rules that could lead to the failure of this constraint. In our example we have two subsets: rules 3 and 4, and 10 and 11 (see Listing 3). The constraint fails because the entity is annotated with both classes.

3.2. Rules and ontology definitions refinement

This step uses the knowledge about the inconsistencies to refine the rules and ontology definitions via three steps: rules clustering, rules and ontology terms ranking, and rules and ontology definitions refinement. These steps are different from our previous work: in our previous work we use an automatic approach that applies a set of predefined refinements to the rules (see Section 2.3.2, steps 2A and 2B in Fig. 3).

3.2.1. Rules clustering

The involved rules are clustered per entity: the entity to which a rule relates is identified, followed by a grouping of the rules per entity. This is done because rules that are related to the same entity have an impact on each other, i.e., other rules might affect the refinement that needs to be applied to a rule.

Example Rules 3-7 and 10-11 are involved in inconsistencies (see Listing 3). Rules 3-7 are related to the entity that is described in Table 1. Rules 10 and 11 are related to the entity that is described in Table 2. This results in two clusters: one with rules 3-7 and one with rules 10-11.

3.2.2. Rules and ontology term ranking

Once the rules are clustered, we rank both the rules and ontology terms to determine th order in which they should be inspected. Thus, we need to define a score to allow such ranking, similar to the score used in the data-driven analysis of DBpedia (see Section 2.3.3).

$$score_c(c) = \frac{|I_c|}{|I|} \tag{1}$$

$$score_t(t) = \frac{|I_t|}{|I|} \tag{2}$$

The scores are calculated based on the inconsistencies in which the rules clusters and ontology terms are involved. R is the set of all rules. C is the set of all rules clusters. T is the set of all ontology terms. I is

the set of all inconsistencies. I_r is the set of inconsistencies in which rule r is involved and $r \in R$. I_c is the set of inconsistencies in which cluster c is involved, i.e., $I_c = \{i | \exists r : (r \in c \wedge i \in I_r)\}$. I_t is the set of inconsistencies in which ontology term t is involved.

The score of a rules cluster c is defined as $score_c(c)$ (see Eq. (1)). It is calculated as the number of inconsistencies a cluster is involved in over the total number of inconsistencies. The score of an ontology term t is defined as $score_t(t)$ (see Eq. (2)). It is calculated as the number of inconsistencies an ontology term is involved in over the total number of inconsistencies. Both scores increase if the number of inconsistencies the rules clusters and ontology terms are involved increases. As a result, they will be ranked higher and inspected earlier by experts. If two scores are equal, then determining which cluster or term is ranked higher happens arbitrarily.

Example We calculate the scores for the two clusters and the four ontology terms. For a cluster, we count the unique inconsistencies in which its rules are involved. The first cluster contains the rules 3-7. Therefore, the number of inconsistencies is 3, resulting in a $score_c$ of 0.75, as there are 4 inconsistencies in total. The second cluster contains the rules 10 and 11. Therefore, the number of inconsistencies is 1, resulting in a $score_c$ of 0.25. This means that the first cluster will be ranked before the second one. Thus, experts will first inspect rules 3-7, because they are together involved in the most inconsistencies and are related to the same entity. The four ontology terms are `ex:Furniture`, `ex:Person`, `ex:madeIn`, and `xsd:string` (see lines 1-3, 6, and 8 in Listing 2). `ex:madeIn` and `xsd:string` are only involved in one inconsistency. Therefore, their corresponding $score_t$ is 0.25. `ex:Furniture` and `ex:Person` are involved in two inconsistencies. Therefore, their corresponding $score_t$ is 0.50. This means that both classes will be ranked before `ex:madeIn` and `xsd:string`. Note that `ex:Furniture` can be ranked before `ex:Person` and vice versa, as they have the same score. Thus, experts will first inspect the definitions of the classes and then the predicate and datatype.

3.2.3. Rules and ontology definitions refinement

Once the rules are ranked, we select the top rules clusters and ontology terms for inspection and determine which refinements should be applied, if any. The inspection is done manually, because the methodology is designed to help experts in determining the desired refinements. Afterwards the rules and ontology definitions can be validated again to detect remaining or newly introduced inconsistencies, which restarts the methodology.

Example Assuming that we only inspect the top rules cluster and ontology term. We start with the rules, followed by the terms. We inspect rules 3 and 4 and observe that the table describes people. Thus, we remove rule 3 and leave rule 4 unchanged. We inspect rules 5 and 6 and observe that the columns "name" and "country" describe the expected names and countries of people. We inspect rule 7 and observe that the used datatype is incorrect: `xsd:string` should be used instead of `ex:Integer`, because the predicate `ex:madeIn` expects strings instead of integers. We inspect `ex:Furniture` and its corresponding definitions 1 and 3 in the ontology. They state that furniture is a class and people and furniture are disjoint. This is still correct and, thus, we leave the definition unchanged.

Once the refinements are applied, we restart the methodology. We notice that there are still inconsistencies present, because we did not resolve all of them correctly during the first iteration. For example, `ex:madeIn` is used with the class `ex:Person`, while the latter is not in the former's domain. This can be resolved by applying another iteration of the methodology's steps.

3.3. Knowledge graph generation

The knowledge graph is generated by applying the semantic annotations to existing data sources via rules (see step 3 in Fig. 3). This graph does not contain the inconsistencies resolved in the previous steps, because the refined rules and ontology definitions are used.

```
1  ex:0 a ex:Person.
2  ex:0 ex:name "Ash".
3  ex:1 ex:madeIn "Belgium".
4  ex:1 a ex:Person.
5  ex:1 ex:name "Misty".
6  ex:1 ex:madeIn "France".
7  ex:21 a ex:Furniture.
8  ex:21 a ex:Person.
9  ex:22 a ex:Furniture.
10 ex:22 a ex:Person.
```

Listing 4: Linked Data generated by applying the refined rules on the data in Tables 1 and 2

Example We generate the knowledge graph based on the refined rules and ontology definitions (see Listing 4). The triples stating that people were also furniture are removed, in accordance with the refinements applied to the rules done in the previous step.

3.4. Knowledge graph inconsistency detection

The knowledge graph is validated to determine inconsistencies (see step 4 in Fig. 3). This is analogous to our previous work and can be done via a number of methods, such as the comparison of the results of queries and inference rules (see Section 2). Inconsistencies that could not be detected using the rules can be detected in this step.

Example Triples 2 and 4 in Listing 4 state the name of a person and cause two inconsistencies. The ontology definitions require the name to be uppercase (see line 11 in Listing 2), but this is not the case, leading to an inconsistency.

3.5. Rules and ontology definitions refinement

Inconsistencies detected in the previous steps might be resolved by refining the rules and ontology definitions (see step 5B in Fig. 3). This is different from the last step of our previous work, where only the rules are refined (see step 5A in Fig. 3).

Example Rules can be added to transform the names of the people to uppercase for use with `ex:name` instead of the original, unchanged values.

4. Implementation

In this work, we use the RDF Mapping Language (RML) [11] as the underlying knowledge graph generation language to apply our methodology (see Section 4.1), because (i) it is an extension of R2RML [8], which is the only W3C standardized knowledge graph generation language (see Section 2.1); (ii) it was used in our previous work [12] (see Section 2.3.2); and (iii) it was used during the data-driven analysis of DBpedia [21] (see Section 2.3.3). We describe the implementation of the methodology's steps (see Sections 4.2, 4.3.1, 4.4 and 4.5). Note that we only discuss the first four steps, because the other steps are analogue to the methodology of our previous work, and not the ontology definitions in steps 3 and 4, because these are independent of the rules and, thus, the language. The complete implementation is available at https://github.com/RMLio/rule-driven-resolution.

4.1. RDF Mapping Language (RML)

RML is a declarative language to define how RDF graphs are generated from existing data sources through a set of rules. RML, as opposed to R2RML [8], does not only support relational databases, but also CSV files, JSON files, XML files, Web APIs, and so on. Furthermore, it is extensible, i.e., data sources in other data formats can be supported in the future. In what follows we describe the details of the language that are relevant for this work. For the full specification, we refer to http://rml.io/spec.html. The corresponding RML rules for our example are given in Listing 5.

```
<#TriplesMap1> a rr:TriplesMap;
  rr:subjectMap <#SM1>;
  rr:predicateObjectMap <#POM1>, <#POM2>,
    <#POM3>, <#POM4>;

  rml:logicalSource [
    a rml:LogicalSource;
    rml:source "table1.csv";
    rml:referenceFormulation ql:CSV ].

<#SM1> a rr:SubjectMap;
  rr:template "http://example.com/{id}".

<#POM1> a rr:PredicateObjectMap;
  rr:predicate rdf:type;
  rr:objectMap <#OM1>.

<#OM1> a rr:ObjectMap;
  rr:constant ex:Furniture.

<#POM2> a rr:PredicateObjectMap;
  rr:predicate rdf:type;
  rr:objectMap <#OM4>.

<#OM4> a rr:ObjectMap;
  rr:constant ex:Furniture.

<#POM3> a rr:PredicateObjectMap;
    rr:predicate ex:name;
    rr:objectMap [
      a rr:ObjectMap;
      rml:reference "name" ].

<#POM4> a rr:PredicateObjectMap;
    rr:predicateMap <#PM1>;
    rr:objectMap [
      a rr:ObjectMap;
      rr:datatype xsd:Integer;
      rml:reference "country" ] ].
```

```
<#PM1> a rr:PredicateMap;
  rr:constant ex:madeIn.

<#TriplesMap2> a rr:TriplesMap;
  rr:predicateObjectMap <#POM5>, <#POM6>;

  rml:logicalSource [
    a rml:LogicalSource;
    rml:source "table2.csv";
    rml:referenceFormulation ql:CSV ];

  rr:subjectMap [
    a rr:SubjectMap;
    rr:template "http://example.com/{id}"
  ].

<#POM5> a rr:PredicateObjectMap;
  rr:predicate rdf:type;
  rr:objectMap <#OM2> .

<#POM6> a rr:PredicateObjectMap;
  rr:predicate rdf:type;
  rr:objectMap <#OM3> .

<#OM2> a rr:ObjectMap;
  rr:constant ex:Furniture.

<#OM3> a rr:ObjectMap;
  rr:constant ex:Person.
```

Listing 5: Example RML rules

For every entity there is a corresponding Triples Map (`rr:TriplesMap`): `<#TriplesMap1>` for the entity in Table 1 and `<#TriplesMap2>` for Table 2. The logical source (`rml:logicalSource`) defines how an existing data source is accessed. In our example, the data are in the files "table1.csv" and "table2.csv" (`rml:source`), in CSV format (`rml:referenceFormulation`). The Term Maps define how the subjects, predicates, and objects of the triples are generated: Subject Map, Predicate Object Map, Predicate Map, and Object Map.

The Subject Map (`rr:SubjectMap`) defines how IRIs are generated via a template (`rr:template`) and form the RDF triples subjects. The "`http://example.com/`", following our example, is concatenated with entity's id.

The Predicate Object Maps define how the triples' predicates and objects are generated; each one requires at least one Predicate and Object Map. In our example, for `<#TriplesMap1>` we have four Predicate Object Maps: name, country, and two for the classes. For the Predicate Object Map that define the classes, we use the `rdf:type` as the predicate (`rr:predicate`). Note that the class does not depend on the data. Therefore, it is constant (`rr:constant` in the Object Map). For the Predicate Object Maps that annotate the entity with its name and place where it is made, we use values from the data (`rml:reference` in the Object Map) instead of constant.

4.2. RML rules inconsistency detection

The inconsistencies in RML rules are detected via a rule-based reasoning system [3]. For each constraint type the corresponding inference rules are created. The RML rules, ontologies, and inference rules serve as the reasoning system's input. The output is inconsistencies with references to the involved RML rules, ontology terms and constraint types.

We rely on a rule-based reasoning system [3], instead of an approach where the results of queries are compared (see Section 2.2), which we used in our previous work, for three reasons: (i) supporting [R2]RML shortcuts via a custom entailment regime, (ii) finding implicit inconsistencies, and (iii) determining the root cause. [R2]RML has defined many shortcuts to make it easier for humans to write [R2]RML rules; however, this results in the fact that different rule sets can result in the same generated RDF dataset [8]. When using a queries execution approach, every shortcut needs to be defined separately per constraint type. By enabling a custom entailment regime via rule-based reasoning, this allows us to define every existing [R2]RML shortcuts as a single custom entailment regime that can be reused for different constraint types. Moreover, we can detect implicit inconsistencies if needed by including another entailment regime [6]. Furthermore, due to the formal logical framework of this reasoning system, we can precisely determine the root causes of the individual inconsistencies using the formal proof, even when including custom entailment regimes. When using an approach where the results of queries are compared, we need a separate, different system to reason over the custom [R2]RML entailment regime. The connection with the original rule set is lost, and the original root cause cannot be found. As correctly identified root causes across different rule sets are an important part of our methodology. ~~We rely on a rule-based reasoning system instead of previously used approach.~~

Example Consider the axiom on line 3 in the ontology (see Listing 2). The corresponding instantiated constraint is as follows: for every combination of Term Maps (Predicate Object Map, Predicate Map, and Object Map) that annotates an entity with the class `ex:Furniture` or `ex:Person`, there should not exist a combination of Term Maps that annotates that same entity with the class `ex:Person` or `Furniture`, respectively. Next, we determine all groups of Term Maps that could lead to the failure of this constraint. In our example we have two groups: `<#POM1>` and `<#POM2>`, and `<#POM5>` and `<#POM6>`. The constraint fails as the entity is annotated with both classes.

4.3. RML rules and ontology definitions refinement

4.3.1. RML rules clustering

The RML rules are clustered by determining the Triples Map to which the rules, i.e., Term Maps, correspond, as every entity is represented by a Triples Map in the rules and every Term Map is related to at least one Triples Map. If the rule is a Predicate Object Map we find the corresponding Triples Map via the predicate `rr:predicateObjectMap` that defines the relationship between the former and the latter. If the rule is a Subject Map we find the corresponding Triples Map via the predicate `rr:subjectMap`. If the rules is a Predicate Map or an Object Map we find the corresponding Triples Map by first finding the corresponding Predicate Object Map via the predicate `rr:predicateMap` and `rr:objectMap`, respectively. Next, the corresponding Triples Map of the found Predicate Object Map is found as described earlier. If the rule is a Triples Map, then the corresponding Triples Map is the Triples Map itself.

Example Lines 11 and 12 contain the RML rules of the Subject Map (`<#SM1>`) of the entities of Table 1. Line 2 connects this Subject Map to its corresponding Triples Map `<#TriplesMap1>` via `rr:subjectMap`. Lines 14 to 16 contain the RML rules of one of the Predicate Object Maps (`<#pom1>`). Line 3 connects this Predicate Object Map to its corresponding Triples Map via `rr:predicateObjectMap`. Lines 18 and 19 contain the RML rules of one of the Object Maps (`<#OM1>`). Line 16 connects this Object Map to its corresponding Predicate Object Map via `rr:objectMap`, which is `<#POM1>`. In turn, this Term Map is used to find the corresponding Triples Map for the Object Map, which is `<#TriplesMap1>`. Lines 41 and 42 contain the RML rules of one of the Predicate Maps (`<PM1>`). Line 35 connects this Predicate Map to its corresponding Predicate Object Map via `rr:predicateMap`, which is `<#POM1>`. In turn, this Term Map is used to find the corresponding Triples Map for the Predicate Map, which is `<#TriplesMap1>`.

If we determine the clusters of the Term Maps `<#POM1>`, `<#PM1>`, `<#OM1>`, l`<#OM2>`, and `<#OM3>`, then we have two clusters. The first cluster corresponds with the Triples Map `<#TriplesMap1>`, and contains `<#POM1>`, `<#PM1>`, and `<#OM1>`. The second cluster corresponds with the Triples Map `<#TriplesMap2>`, and contains `<#OM2>` and `<#OM3>`.

4.4. RML rules ranking

We calculate the score of every rules cluster and ontology term to be able to rank them. We iterate over each cluster, which is identified by the Triples Map that represents an entity. We iterate over every Terms Map that is in the cluster and count the inconsistencies in which it is involved. Note that for a single cluster we count an inconsistency only once, even if two Term Maps are involved in the same inconsistency. The score equals this count over the total number of inconsistencies (see Eq. (1)). Furthermore, the ranking of the ontology definitions does not depend on the used language. Thus, it is done as described in Section 3.2.2.

Example The cluster of `<#TriplesMap2>` is involved in one inconsistency of the four inconsistencies detected. `<#OM2>` and `<#OM3>` annotate the entity with the classes `ex:Furniture` and `ex:Person`, which leads to an inconsistency as these two classes are disjoint (see Listing 2). Thus, the score of the cluster is 0.25, analogues to our example in Section 3.2.2.

4.5. RML rules refinement

Once the ranking is done, experts inspect the top rules clusters, identified by the Triples Maps, and apply the necessary refinements to the RML rules. Note that the refinement of the ontology definitions does not depend on the used language. Thus, it is done as described in Section 3.2.3.

Example Let's assume that we only inspect the top rules cluster. We inspect Predicate Object Maps `<#POM1>` and `<#POM2>` , including their corresponding Predicate and Object Maps and see that the table describes people. Thus, we remove `<#POM1>` and the corresponding Term Maps, and leave `<#POM2>` unchanged. We inspect `<#POM3>` and `<#POM4>` and see that the columns "name" and "country" describe the expected names and countries of people. We inspect inspect the Object Map of `<#POM4>` and see that the used datatype is incorrect: `xsd:string` should be used instead of `ex:Integer`, because the predicate `ex:madeIn` expects strings instead of integers.

Once the refinements are applied, we restart the methodology. We notice that there are still inconsistencies present, because we did not resolve all of them correctly during the first iteration. For example, `ex:madeIn` is used with the class `ex:Person`, while the latter is not in the former's domain. This can be resolved by applying another iteration of the methodology's steps.

4.6. Knowledge graph generation

The knowledge graph is generated by applying the semantic annotations to the existing data sources via the RML rules. This graph does not contain the inconsistencies resolved in the previous steps, because the refined RML rules and ontology definitions are used (see Listing 4).

4.7. Knowledge graph inconsistency detection

The knowledge graph, which is an RDF graph, is validated to determine inconsistencies. Inconsistencies that could not be detected using rules can be detected in this step. More, this step is independent of the used language, i.e., RML, because only the knowledge graph is used and not the rules.

Example Triples 2 and 4 in Listing 4 state the name of a person and cause two inconsistencies. The ontology definitions require the name to be uppercase (see line 11 in Listing 2), but this is not the case, leading to an inconsistency.

4.8. RML rules refinement

Inconsistencies detected in the previous steps might be resolved by refining the RML rules and ontology definitions.

Example RML rules can be added to transform the names of the people to uppercase for use with `ex:name` instead of the original, unchanged values. This is done by using a function that returns the uppercase version of a string. We use the Function ontology [9] to declarative describe this transformation in the RML rules [10]. We refine lines 30 and 32: they are removed and replaced by `rr:objectMap <#ToUpperCase>`, and we add the following rules:

```
<#ToUpperCase>
  fnml:functionValue [
    rr:predicateObjectMap [
      rr:predicate fno:executes ;
      rr:objectMap [
        rr:constant grel:toUppercase
      ]
    ] ;

    rr:predicateObjectMap [
      rr:predicate grel:inputString ;
      rr:objectMap [
        rr:reference "name"
      ]
    ]
  ].
```

The data in column "name" is used as input for the function `grel:toUppercase`[2], which returns the uppercase version of a string. For a detailed description about functions and these rules, we refer to https://w3id.org/function/.

5. Findings

We apply our methodology to two real-life use cases: DBpedia [5] (see Section 5.1) and the Computer Science bibliography[3] (see Section 5.2). We analyze and discuss the inconsistencies, involved rules, and ontology definitions. More specific, we investigate the difference between the use of clusters when ranking and refining the rules and the use of the rules directly, i.e., without clustering. The calculation of the rules' scores without clustering is analogous to the ontology terms' scores.

5.1. DBpedia

The DBpedia knowledge graph is generated based on the information in the Wikipedia infoboxes. Originally, a wiki markup syntax for the rules was used. However, this syntax does not allow the rules to be validated directly, and thus certainly not in the same way as the complete knowledge graph. In our previous work [12], we automated the conversion of all original rules to RML rules[4] to make them processable by our methodology.

5.1.1. Quantitative results

In total there are 1,212,337 rules and 2,159 inconsistencies, where 1,370 rules and 8 ontology definitions contribute to at least one inconsistency. The rules are related to 398 entities, resulting in the same number of clusters (see Table 4).

5.1.2. Analysis

Rules We inspect the top rules cluster which is about libraries, contains 6 rules, and is involved in 4 inconsistencies. The inconsistencies are caused by the combination of the used classes and properties, and missing rules that define the datatypes of objects (see Table 5). The classes that are defined for a library are closely related to the used properties. For example, `dbo:Organisation`, `dbo:Location`, `dbo:Place`, and `schema:Place` are a number of the classes of a library, and `foaf:name`, `geo:long`, and `geo:lat` are a number of the properties. The classes `dbo:Location` and `schema:Place` state that a library is a location on earth, and the properties `geo:long` and `geo:lat` provide details about the position of the library.

When comparing the use of clustering during the ranking and refinement to the use of no clustering, we notice that the rules that define the classes and the properties of a library are not grouped together (see Table 6). For example, the rule that defines the classes is the top rule, while the other rules are ranked on the 36th, 37th, 366th, 891th, and 892th position (see https://doi.org/10.6084/m9.figshare.6834005.v1 for details). This happens because the rule that defines classes is involved in all consistencies where the restrictions on domains of properties are not respected, while the rules that define these properties are only involved in that one specific inconsistency that represents the domain restriction of a specific property. Thus, without clustering, experts would refine the classes without taking into account the predicates defined by the rules, as they do not appear in the top rules. For example, the top rules of a library do not give the necessary insights about the rules that determine the predicates, which is important to resolve inconsistencies that are related to domain restrictions, such those of `geo:long` and `foaf:name`.

[2] `grel` is the prefix for the namespace http://semweb.datasciencelab.be/ns/grel#

[3] https://dblp.uni-trier.de/

[4] https://doi.org/10.6084/m9.figshare.6834050.v1

use case	# rules	# inconsistencies	# involved rules	# clusters	# involved definitions
DBpedia	1,212,337	2,159	698	398	8
DBLP	368	12	18	5	8

Table 3
Findings of applying our methodology to DBpedia and DBLP

entity	# rules	score
library	6	0.018
region of Italy	13	0.013
body of water	11	0.013
Australian place	10	0.013
university	7	0.013
Singapore school	7	0.013
komunne	10	0.012
Greek isles	10	0.012
savivaldybe	10	0.012
French region	10	0.012

Table 4
10 top rules clusters for DBpedia

Ontology terms As the rules use ontology terms to annotate the data, these terms are also involved with the inconsistencies and require inspection. We inspect the ontology definitions that are involved in at least one inconsistency (see Table 7). The definitions are part of two ontologies: WGS84 Geo Position ontology[5] and the Friend of a Friend (FOAF) project[6]. `geo:SpacialThing` is the expected domain of `geo:long` and `geo:lat`; `foaf:Person` of `foaf:name`, `foaf:surname`, and `foaf:familyName`; and `foaf:Image` of `foaf:thumbnail`. Therefore, inconsistencies are introduced in the rules when these predicates are used with entities that are not annotated with these specific classes. For example, these inconsistencies are present for the library: the use of `geo:long` and `geo:lat` without `geo:SpatialThing`, and `foaf:name` without `foaf:Person`.

Refinements It is possible to refine the classes of entities to resolve inconsistencies that are due to domains of predicates. It might be possible that the domains of predicates are not correct with respect to the desired model by the user. Therefore, changes to the domains might be required to resolve the inconsistencies instead of refining the rules.

In the case of libraries, we can consider a library a spacial thing that has longitude and latitude information. Thus, a possible refinement is adding a rule that annotate a library with the class `geo:SpacialThing`. However, a library should probably not be considered a person via `foaf:Person`, which is the case when using the predicate `foaf:name` to describe the name of a library. Considering that the FOAF ontology is concerned with describing information about people, changing it is not appropriate. Thus, the rules should be refined by using definitions from a different ontology or by removing the rule that annotates the data with `foaf:name`. For example, the predicates `rdfs:label` and `dbo:name` could be used instead of `foaf:name`. Both predicates do not have a restriction on the domain.

5.2. The Computer Science bibliography

The Computer Science bibliography (DBLP) collects open bibliographic information from major computer science journals and proceedings. DBLP rules are originally defined using D2RQ and were converted to RML[7] using D2RQ-to-R2RML[8] to be processed by our implementation.

5.2.1. Quantitative results

In total, there are 368 rules and 12 inconsistencies, where 14 rules and 5 ontology definitions contribute to at least one inconsistency. The rules are related to 5 entities, resulting in the same number of clusters (see Table 8).

5.2.2. Analysis

We inspect the top rules cluster which is about publications, contains 6 rules, and is involved in 5 inconsistencies. The causes for the inconsistencies are the combinations of the used classes and properties, and datatypes and properties. The classes that are defined for a publication are closely related to the used properties. For example, the only class of a publication is `foaf:Document` and a number of the properties are `dcterms:bibliographicCitation`, `foaf:page`, `swrc:editor`, and `dc:publisher`. The same holds for the properties and datatypes. For example, the data property `dcterms:bibliographicCitation` expects a literal as object.

When comparing the use of clustering during the ranking and refinement to the use of no clustering, we notice that the rules that define the classes and the properties of a publication are not grouped together (see Table 9). For example, the rule that define the classes is the top rule, while 4 out of the 5 other rules regarding publications are ranked lowest (see https://doi.org/10.6084/m9.figshare.6834107.v1 for

[5] prefix: `geo`; http://www.w3.org/2003/01/geo/wgs84_pos#
[6] prefix: `foaf`; http://xmlns.com/foaf/spec/
[7] https://doi.org/10.6084/m9.figshare.6834065.v1
[8] https://github.com/RMLio/D2RQ_to_R2RML.git

rule	involved definitions
lib:SubjectMap, lib:longitude-128525	class of publication is not in domain of predicate `geo:long`
lib:SubjectMap, lib:latitude-128115	class of publication is not in domain of predicate `geo:lat`
lib:SubjectMap, lib2:name_en-125405	class of publication is not in domain of predicate `foaf:name`
lib2:name_en/FunctionTermMap, lib2:library_name/FunctionTermMap, lib:	predicate `foaf:name` is datatype property, but no datatype provided

Table 5

Alignment of the rules of the top cluster and the involved ontology definitions (`lib` is the prefix for the namespace http://en.dbpedia.org/resource/Mapping_en:Infobox_library/ and `lib2` for http://en.dbpedia.org/resource/Mapping_en:Infobox_library/SimplePropertyMapping/foaf:name/)

rule	score
Infobox_library/Infobox_library/SubjectMap	0.47
Infobox_education_in_Canada/Infobox_education_in_Canada/SubjectMap	0.42
Infobox_Singapore_school/Infobox_Singapore_school/SubjectMap	0.42
Infobox_UK_school/Infobox_UK_school/SubjectMap	0.42
Infobox_university/Infobox_university/SubjectMap	0.42
Infobox_Australian_place/Infobox_Australian_place/SubjectMap	0.41
Infobox_body_of_water/Infobox_body_of_water/SubjectMap	0.41
Infobox_broadcast/Infobox_broadcast/SubjectMap	0.41
Infobox_PNG_Place/Infobox_PNG_Place/SubjectMap	0.41
Infobox_radio_station/Infobox_radio_station/SubjectMap	0.41

Table 6

10 top rules for DBpedia

ontology term	score
geo:SpacialThing	0.676
geo:long	0.338
geo:lat	0.338
foaf:name	0.310
foaf:Person	0.011
foaf:surname	0.006
foaf:familyName	0.006
foaf:thumbnail	0.002

Table 7

8 involved ontology terms for DBpedia

details). Thus, without clustering experts would refine the classes without taking into account the predicates that are defined by the rules, as they do not appear in the top rules. This is analogous to the DBpedia use case.

Ontology terms As the rules use ontology terms to annotate the data, they are also involved with the inconsistencies and require inspection. We inspect the ontology definitions that contribute to at least one inconsistency (see Table 11). The definitions are part of five ontologies: RDF Schema[9], XML Schema[10], DC Terms[11], DC Elements[12], and The Semantic Web for Research Communities ontology[13]. 4 of the 5 ontologies are involved in the inconsistencies of the top cluster: XML Schema, DC Terms, DC Elements, and the Semantic Web for Research Communities ontology. More specific, 4 out of 5 ontology terms are involved: `xsd:string`, `dcterms:bibliographicCitation`, `dc:publisher`, and `swrc:editor`.

entity	# rules	score
publications	6	0.5
authors	2	0.2
journals	2	0.1
conferences	2	0.1
collections	2	0.1

Table 8

5 rules clusters for DBLP

[9] prefix: `rdfs`; http://www.w3.org/2000/01/rdf-schema#
[10] prefix: `xsd`; http://www.w3.org/2001/XMLSchema#
[11] prefix: `dcterms`; http://purl.org/dc/terms/
[12] prefix: `dc`; http://purl.org/dc/elements/1.1/
[13] prefix: `swrc`; http://swrc.ontoware.org/ontology#

entity	score
Publications_subjectMap	0.54
authorName_ObjMap	0.43
Authors	0.43
publicationPublisher-58	0.43
CollectionName_ObjMap	0.27
CollectionName-28	0.27
ConferenceName_ObjMap	0.27
ConferenceName-27	0.27
JournalName_ObjMap	0.27
JournalName-26	0.27

Table 9
Top 10 rules for DBLP

The use of the different ontologies and the corresponding predicates might explain the detected inconsistencies with the domain, because the publications need to be annotated with the different classes from the different ontologies, which might not be taken into consideration when defining the rules. The inconsistencies that involve datatype properties expect a different datatype than the one used in the rules.

Refinements It is possible to refine the classes of entities to resolve inconsistencies that are due to domains of predicates. It might be possible that the domains of predicates are not correct with respect to the desired model by the user. Therefore, changes to the domains might be required to resolve the inconsistencies instead of refining the rules.

The classes that are missing according to the inconsistencies are `dcterms:BibliographicResource` and `swrc:Proceedings`. A publication is a bibliographic resource, but not every publication is proceedings. However, `swrc:editor` only has `swrc:Proceedings` in its domain and not `swrc:Proceedings`. On the one hand, we can refine the rules and also model the proceedings if the data is available. This allows us to add the information about the editors to the these proceedings instead of the publications. This leaves the ontology unchanged; however, if editors are related to a specific publication and not the proceedings that contain it, then the refined rules and corresponding knowledge graph will not reflect this. On the other hand, we can refine the ontology definitions and add `swrc:Publication` to the domain of `swrc:editor`. This leaves the rules unchanged; however, this might make the ontology use case specific when the fact that editors are related to a specific publication is specific to the Computer Science bibliography. The predicate `dc:publisher` has as range `dcterms:Agent`, but the rules generate a literal with the datatype `xsd:string`. On the one hand, we can refine the rules and also model the publishers as entities if the data is available, analogous to the proceedings. This allows us to annotate the publishers as agents, which is in the range of the predicate. On the other hand, we can refine the ontology definitions by removing the restriction on the range. This will not lead to new inconsistencies in existing knowledge graphs, but might lead to less structured knowledge as all the information of a publisher can be provided in a single string.

6. Conclusion

Knowledge graphs are often generated by applying generation rules that annotate data in existing data sources with ontologies terms. However, the knowledge graphs might suffer from inconsistencies, which can be introduced by the combination of rules and ontologies. These inconsistencies can be resolved by either refining the rules or ontologies. In this article, we introduce a rule-driven methodology that ranks the rules, via clustering, and ontology definitions involved in an inconsistency. The top rules and ontology definitions are inspected by experts and the necessary refinements are applied to resolve the inconsistencies.

The findings of the real-life use cases show that the rules clustering allows inspecting rules related to the same entity. Therefore, experts have better insights to apply the desired refinements, because they can immediately inspect all rules of that specific entity that is involved in an inconsistency.

Refinements can be applied to both rules and ontology definitions to resolve the inconsistencies. Nevertheless, experts need to carefully determine whether the former or latter needs to be refined. For example, is it desired to refine an ontology that models the data of a specific use case? Is it desired to keep to the ontology as it is and refine the rules? Or should another ontology be used instead of the current one? Our methodology, including the rankings, provides valuable insights to answer these questions, such as the entities that need to most attention when applying refinements, and the specific ontology terms and definitions that are involved in a lot of inconsistencies and, thus, might be problematic.

The ranking proposed in our methodology can be used not only by experts to explore manual refinements, but can also be used to drive (semi-)automatic solution. For example, this can be done by building on our previous work were we proposed an automatic solution that resolve the inconsistencies by applying a predefined set of refinements to the rules. The ranking, which is use case-specific, can be used to make a more informed decision regarding which refinements should be applied, instead of solely relying on a predefined set, which is created independent of the use case.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Research Foundation – Flanders (FWO), and the European Union.

rule	involved definitions
publicationBibTeX-48, Publications_subjectMap	class of publication is not in domain of predicate `dcterms:bibliographicCitation`
publicationEditor-52, Publications_subjectMap	class of publication is not in domain of predicate `swrc:editor`
publicationPublisher_ObjMap, publicationPublisher-58	datatype `xsd:string` of object is not as expected by predicate `dc:publisher`
publicationPublisher-58, Publications_subjectMap	class of publication is not in domain of predicate `dc:publisher`
publicationBibTeX_ObjectMap	predicate `dcterms:bibliographicCitation` is datatype property, but no datatype is provided

Table 10
Findings of applying our methodology to DBpedia and DBLP

ontology term	score
rdfs:label	0.5
xsd:string	0.4
dcterms:bibliographicCitation	0.2
dc:publisher	0.2
swrc:editor	0.1

Table 11
8 involved ontology terms for DBLP

References

[1] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing Linked Data Quality Assessment. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, pages 260–276, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[2] Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann Publishers Inc., 2 edition, 2011. ISBN 9780123859655, 9780123859662.

[3] Dörthe Arndt, Ben De Meester, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. Using Rule-Based Reasoning for RDF Validation. In Stefania Costantini, Enrico Franconi, William Van Woensel, Roman Kontchakov, Fariba Sadri, and Dumitru Roman, editors, *Proceedings of the International Joint Conference on Rules and Reasoning*, volume 10364 of *Lecture Notes in Computer Science*, pages 22–36. Springer, July 2017. ISBN 978-3-319-61252-2. .

[4] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data: The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):205–227, 2009. .

[5] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009. ISSN 1570-8268. .

[6] Thomas Bosch, Andreas Nolle, Erman Acar, and Kai Eckert. RDF Validation Requirements – Evaluation and Logical Underpinning. *arXiv preprint arXiv:1501.03933*, 2015. URL http://arxiv.org/abs/1501.03933.

[7] World Wide Web Consortium. RDF 1.1 Concepts and Abstract Syntax. Technical report, 2014. URL https://www.w3.org/TR/rdf11-concepts/.

[8] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. Working group recommendation, W3C, September 2012. URL http://www.w3.org/TR/r2rml/.

[9] Ben De Meester, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. An ontology to semantically declare and describe functions. In Harald Sack, Giuseppe Rizzo, Nadine Steinmetz, Dunja Mladenić, Sören Auer, and Christoph Lange, editors, *The Semantic Web; ESWC 2016 Satellite Events*, volume 9989 of *Lecture Notes in Computer Science*, pages 46–49. Springer International Publishing, October 2016. .

[10] Ben De Meester, Wouter Maroy, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. Declarative data transformations for Linked Data generation: The case of DBpedia. In Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *Proceedings of the 14th ESWC*, volume 10250 of *Lecture Notes in Computer Science*, pages 33–48. Springer, May 2017. ISBN 978-3-319-58451-5. .

[11] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Workshop on Linked Data on the Web*, 2014.

[12] Anastasia Dimou, Dimitris Kontokostas, Markus Freudenberg, Ruben Verborgh, Jens Lehmann, Erik Mannens, Sebastian Hellmann, and Rik Van de Walle. Assessing and refining mappings to RDF to improve dataset quality. In *Proceedings of the 14th International Semantic Web Conference*, volume 9367 of *Lecture Notes in Computer Science*, pages 133–149. Springer, October 2015.

[13] Anastasia Dimou, Dimitris Kontokostas, Markus Freudenberg, Ruben Verborgh, Jens Lehmann, Erik Mannens, and Sebastian Hellmann. DBpedia mappings quality assessment. In Takahiro Kawamura and Heiko Paulheim, editors, *Proceedings of the 15th International Semantic Web Conference: Posters and Demos*, volume 1690 of *CEUR Workshop Proceedings*, October 2016. URL http://ceur-ws.org/Vol-1690/paper97.pdf.

[14] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase,

opencyc, wikidata, and yago. *Semantic Web*, (Preprint):1–53, 2016.

[15] Mina Farid, Alexandra Roatis, Ihab F. Ilyas, Hella-Franziska Hoffmann, and Xu Chu. CLAMS: Bringing Quality to Data Lakes. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2089–2092. ACM, 2016. ISBN 978-1-4503-3531-7. .

[16] Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL). W3C recommendation, W3C, July 2017. URL https://www.w3.org/TR/shacl/.

[17] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven Evaluation of Linked Data Quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758. ACM, 2014. ISBN 978-1-4503-2744-2. .

[18] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *European Semantic Web Conference*, pages 35–50. Springer, 2017.

[19] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[20] Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. Sieve: Linked Data Quality Assessment and Fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, pages 116–123. ACM, 2012. ISBN 978-1-4503-1143-4. .

[21] Heiko Paulheim. Data-Driven Joint Debugging of the DBpedia Mappings and Ontology. In Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *The Semantic Web*, pages 404–418. Springer International Publishing, 2017.

[22] Heiko Paulheim. Knowledge graph refinement: a survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.