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, which contain annotated descriptions of entities and their interrelations, are often generated using rules that apply semantic annotations to certain data sources. (Re)using ontology terms without adhering to the axioms defined by their ontologies results in inconsistencies in these graphs, affecting their quality. Methods and tools were proposed to detect and resolve inconsistencies, the root causes of which include rules and ontologies. However, these either require access to the complete knowledge graph, which is not always available in a time-constrained situation, or assume that only generation rules can be refined but not ontologies. In the past, we proposed a rule-driven method for detecting and resolving inconsistencies without complete knowledge graph access, but it requires a predefined set of refinements to the rules and does not guide users with respect to the order the rules should be inspected. We extend our previous work with a rule-driven method, called Resglass, that considers refinements for generation rules as well as ontologies. In this article, we describe Resglass, which includes a ranking to determine the order with which rules and ontology elements should be inspected, and its implementation. The ranking is evaluated by comparing the manual ranking of experts to our automatic ranking. The evaluation shows that our automatic ranking achieves an overlap of 80% with experts ranking, reducing this way the effort required during the resolution of inconsistencies in both rules and ontologies.

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

1. Introduction

Knowledge graphs use ontologies to provide annotated descriptions of entities and their interrelations [28]. The graphs can be published as Linked Data [5] using the Resource Description framework (RDF) [7] as data representation format.

Knowledge graphs are often generated from other sources. For instance, the DBpedia knowledge graph is generated from Wikipedia. A common way to generate these knowledge graphs is by using rules. The rules attach semantic annotations to data in those sources. The semantic annotations are added using terms defined in ontologies, such as *classes*, *properties*, and *datatypes*. These rules thereby determine how individual data fragments from the sources are modeled using specific ontology terms during knowledge graph generation. The syntax and grammar of the rules are determined by a knowledge graph generation language, such as R2RML [8], RML [11], and SPARQL-Generate [23].

An *ontology* is a conceptualization, an intensional semantic structure, which encodes the implicit rules constraining the structure of a piece of reality [17]. Such implicit rules can be encoded as ontological *axioms* in OWL [26] and are henceforth referred to as *restrictions*. For example, the domains and ranges of properties are restricted to a set of classes. Such restrictions are either defined via the ontology term's definitions, e.g., a term is a class or a property, or via the interpretation of the ontology's axioms, e.g, domain and range of a property, as restrictions [2, 22].

Inconsistencies are introduced in graphs when ontology terms are used without adhering to restrictions, and this affects the graphs' quality. Possible root causes for these inconsistencies include: (i) *raw data* that contain inconsistencies [24]; (ii) *rules* that intro-

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

duce new inconsistencies by, for example, not using the suitable ontology terms [12, 27]; and (iii) *ontology definitions* that do not model the domain as desired [27]. In this work, we focus on the latter two root causes which are related to the intrinsic dimension of knowledge graph quality [11].

Previous research efforts introduced methods and tools to identify inconsistencies in knowledge graphs [4, 22]. Although this enables resolving the inconsistencies in the graph itself, it does not fix the root cause. The same inconsistencies reappear when a new version of a knowledge graph is generated. Thus, methods and tools were developed to identify inconsistencies in knowledge graph generation rules [12, 27]. Methods applied to generation rules find inconsistencies in less time compared to solutions that work directly on knowledge graphs, while they simultaneously identify the rules and ontology definitions causing them [13]. For instance, there were 2,159 inconsistencies identified in the rules that define how the DBpedia knowledge graph is generated from Wikipedia.

The rules or ontology definitions need to be refined to resolve inconsistencies, but this is not straightforward. The situation aggravates when the set of rules and their relationships grow, or multiple and more complex ontologies are used. For instance, more than 1,300 of the more than 1,200,000 rules that generate DBpedia are involved in at least 1 of the 2,159 identified inconsistencies. Which rules should users inspect first when resolving these inconsistencies, considering that updating a rule can resolve multiple inconsistencies, but it can also create new ones? Furthermore, a number of inconsistencies are best resolved by updating the ontology definitions and not the rules [27]. Should a user inspect the rules or also the ontology definitions, considering that inconsistencies might be caused by both rules and ontology definitions?

We proposed a rule-driven method in previous work to resolve inconsistencies by automatically refining the corresponding generation rules [12]. Inconsistencies are detected by analyzing primarily the rules, but also the knowledge graphs. Predefined refinements are automatically applied to resolve the inconsistencies. However, our rules-driven method assumes that used ontologies align with the user's envisioned semantic model, which is not always the case [27]. More, when a high number of rules are involved in inconsistencies, users have no insights regarding the order with which rules should be inspected. This leads to the following research question to improve the resolution process:

How can we score and rank rules and ontology terms for inspection to improve the manual resolution of inconsistencies?

In this work, we introduce a new method called Resglass that extends our rule-driven method [12] to address this research question. The rules and ontology terms are automatically ranked in order of inspection based on a score that considers the number of inconsistencies a rule or ontology term is involved in. This way, the automatic ranking accelerates the inconsistencies resolution speed, as users do not need manually provide a ranking anymore. Resglass consists of the following steps:

1. (automatically) detect inconsistencies by analyzing generation rules;
2. (automatically) cluster the rules;
3. (automatically) rank rules and ontology terms in order of expected impact upon inspection;
4. (manually) refine rules and ontology terms;
5. (automatically) generate the knowledge graph;
6. (automatically) detect inconsistencies in this graph;
7. (manually) refine rules and ontologies terms.

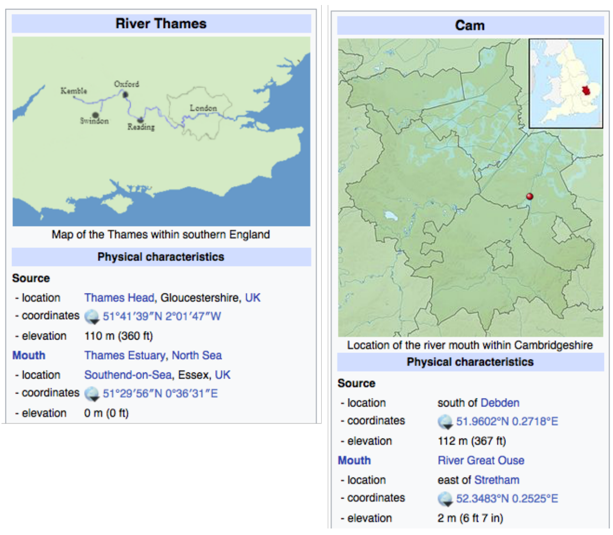
This research question leads to the hypothesis:

An automatic inconsistency-driven ranking improves, compared to a random ranking, by at least 20% the overlap with experts' manual ranking.

Our novel contributions include in particular:

- an algorithm to *rank rules* in descending order of number of inconsistencies they are involved in;
- an algorithm to *rank ontology terms* in descending order of number of inconsistencies they are involved in;
- an *implementation of the Resglass* using an existing knowledge graph generation language, RML;
- an *evaluation* that shows how much our automatic ranking improves the overlap to the experts' ranking, compared to a random ranking,

Our evaluation shows that our ranking improves the overlap with experts' ranking by 40% in the case of rules and 20% in the case of ontology terms. Overall, Resglass reduces the effort required during the resolution of inconsistencies in both rules and ontologies, because less manual effort is required from the experts, leading to higher quality knowledge graphs in less time and with less effort.



River Thames

Map of the Thames within southern England

Physical characteristics

Source	
- location	Thames Head, Gloucestershire, UK
- coordinates	51°41′39″N 2°01′47″W
- elevation	110 m (360 ft)
Mouth	Thames Estuary, North Sea
- location	Southend-on-Sea, Essex, UK
- coordinates	51°29′56″N 0°36′31″E
- elevation	0 m (0 ft)

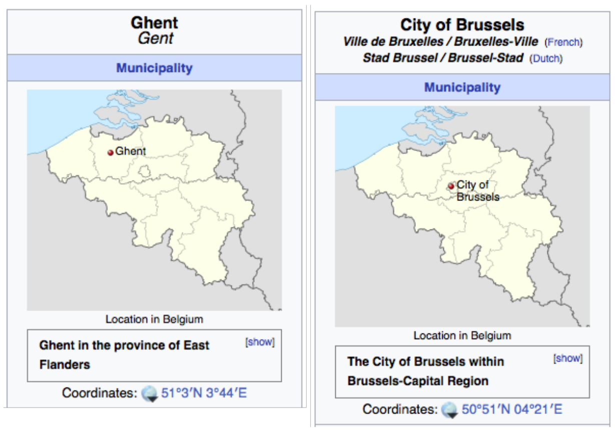
Cam

Location of the river mouth within Cambridgeshire

Physical characteristics

Source	
- location	south of Debden
- coordinates	51.9602°N 0.2718°E
- elevation	112 m (367 ft)
Mouth	River Great Ouse
- location	east of Stretham
- coordinates	52.3483°N 0.2525°E
- elevation	2 m (6 ft 7 in)

Figure 1. River Thames and Cam Wikipedia infobox



Ghent
Gent

Municipality

Location in Belgium

Ghent in the province of East Flanders [show]

Coordinates: 51°3′N 3°44′E

City of Brussels
Ville de Bruxelles / Bruxelles-Ville (French)
Stad Brussel / Brussel-Stad (Dutch)

Municipality

Location in Belgium

The City of Brussels within Brussels-Capital Region [show]

Coordinates: 50°51′N 04°21′E

Figure 2. Ghent and Brussels Wikipedia infobox

The remainder of this article is structured as follows: In Section 2, we introduce our motivating use case. In Section 3, we discuss the state of the art. In Section 4, we elaborate on the Resglass and, in Section 5, on the implementation. In Section 6, we present our evaluation and, in Section 7, our conclusions.

2. Motivating use case

As motivating use case, we consider DBpedia. The DBpedia knowledge graph is generated from Wikipedia. The rules that define how to annotate the Wikipedia infoboxes with classes and properties from different ontologies, including the DBpedia, FOAF and GS84 Geo Positioning ontologies are defined in rules expressed in RML. However, inconsistencies appear in this knowledge graph [22] and are caused by both rules and ontology definitions [27].

The two types of Wikipedia infoboxes in Figs. 1 and 2, provide details of two rivers (Thames and Cam) and two cities (Ghent and Brussels). Each infobox presents the relevant information about a single entity based on a predefined infobox template. For example, the infoboxes used for rivers follow the same template[1]. The rules (Listing 1) annotate the data with ontology definitions (Listing 2) that describe rivers, palaces and cities, and generate the knowledge graph (Listing 3). Inconsistencies can then be detected by comparing the rules and ontologies definitions.

1st inconsistency: The triple in line 2 states that `_:b0` is the source position of river Thames and the triple in line 5 states that `_:b0` is of class `dbo:Place` (see Listing 3). The same holds for river Cam and its source position `_:b1`. However, the ontology defines that the class `geo:SpatialThing`, and not `dbo:Place`, is in the range of the property `dbo:sourcePosition` (see Listing 2).

2nd inconsistency: The triple in line 3 states that `_:b2` is the mouth position of river Thames and the triple in line 8 states that `_:b2` is of class `dbo:Place`. However, the ontology defines that the range of `dbo:mouthPosition` is the class `geo:SpatialThing`.

3rd inconsistency: The triples in line 6 and 16 state `_:b0`'s and `_:b1`'s latitude, respectively. However, the ontology defines that `geo:SpatialThing`, and not `dbo:Place`, is in the domain of `geo:lat`.

The *1st inconsistency* is caused due to the combination of rules 21 and 38 (Listing 1). Rule 21 links rivers with their locations via `dbo:sourcePosition` and rule 38 annotates a location with the class `dbo:Place`. This is not consistent with the ontology, as the class `geo:SpatialThing`, and not `dbo:Place`, is in the range of `dbo:sourcePosition`.

The *2nd inconsistency* is caused due to the combination of rules 29 and 52. Rule 29 links rivers with their locations via `dbo:mouthPosition` and rule 52 annotates a location with the class `dbo:Place`. This is not consistent with the ontology, as class `geo:SpatialThing`, and not `dbo:Place`, is in the range of `dbo:mouthPosition`.

The *3rd inconsistency* is caused due to the combination of rules 38 and 43. Rule 38 annotates a lo-

[1]https://en.wikipedia.org/wiki/Template:Infobox_river

cation with the class `dbo:Place` and rule 43 the latitude of a location with the property `geo:lat`. This is not consistent with the ontology, as class `geo:SpatialThing`, and not `dbo:Place`, is in the domain of `geo:lat`.

How should these inconsistencies be resolved? On the one hand, which rules should be refined? Is it one of the rules 21, 29, 38, 43, 52, or all of them? How should they be refined: should the class in rule 38 be `geo:SpatialThing` or another class? What are the effects of updating these rules? For instance, this refinement would lead to other inconsistencies, such as caused by rules 38 and 66 if the entity is not longer annotated with the class `dbo:Place`. On the other hand, if the ontology can be updated, which definitions should be refined? How should the definitions be refined: remove definitions 7, 9, and 13, or add other definitions to expand the range of `dbo:sourcePosition`, `dbo:mouthPosition`, and the domain of `geo:lat`?

Although we just discussed a small subset of the rules and definitions of DBpedia, we already have a number of different rules and definitions to consider for inspection, and different approaches to resolve the inconsistencies, which might introduce new inconsistencies. This aggravates when we consider the whole of DBpedia where more than 1,300 of the more than 1,200,000 rules are involved in at least one of the 2,159 inconsistencies, and where the DBpedia ontology[2] alone already consists of more than 700 classes and more than 2,800 properties.

```
1  <#TriplesMap1> rr:subjectMap <#SM1>;
2  rr:predicateObjectMap
3  <#POM1>, <#POM2>, <#POM3>.
4
5  <#SM1> rr:template
6  "http://dbpedia.org/resource/{slug}".
7
8  <#POM1> rr:predicate rdf:type;
9  rr:objectMap <#OM1>.
10
11 <#OM1> rr:constant dbo:River.
12
13 <#POM2> rr:predicate foaf:name;
14 rr:objectMap [rml:reference "name"].
15
16 <#POM3> rr:predicateMap <#PM1>;
17 rr:objectMap[rr:joinCondition[
18 rr:child "slug"; rr:parent "slug";
19 rr:parentTriplesMap <#TriplesMap2>]].
20
21 <#PM1> rr:constant dbo:sourcePosition.
22
23 <#POM3> rr:predicateMap <#PM4>;
24 rr:objectMap[rr:joinCondition[
25 rr:child "slug"; rr:parent "slug";
26 rr:parentTriplesMap <#TriplesMap3>]].
27
28 <#PM4> rr:constant
29  dbo:mouthPositionPosition.
30
31 <#TriplesMap2>
32 rr:subjectMap[rr:termType rr:BlankNode].
33 rr:predicateObjectMap <#POM4>,<#POM5>.
34
35 <#POM4> rr:predicate rdf:type;
36 rr:objectMap <#OM2>.
37
38 <#OM2> rr:constant dbo:Place.
39
40 <#POM5> predicateMap <#PM2>;
41 rr:objectMap[rml:reference "latitude"].
42
43 <#PM2> rr:constant geo:lat.
44
45 <#TriplesMap3>
46 rr:subjectMap[rr:termType rr:BlankNode].
47 rr:predicateObjectMap <#POM6>,<#POM7>.
48
49 <#POM6> rr:predicate rdf:type;
50 rr:objectMap <#OM3>.
51
52 <#OM3> rr:constant dbo:Place.
53
54 <#POM7> predicateMap <#PM3>;
55 rr:objectMap[rml:reference "latitude_m"].
56
57 <#PM3> rr:constant geo:lat.
58
59 <#TriplesMap4>
60 rr:predicateObjectMap [
61 rr:predicateMap <#PM5>;
62 rr:objectMap[rr:joinCondition[
63 rr:child "slug"; rr:parent "slug";
64 rr:parentTriplesMap <#TriplesMap2>]].
65
66 <#PM5> rr:constant dbo:location.
```

Listing 1: Subset of example RML rules

1 dbo:River is a class
2 dbo:City is a class
3 dbo:River and dbo:City are disjoint

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

```
dbo:Place is a class
geo:SpatialThing is a class
dbo:location is a property
dbo:Place is in the range of dbo:location
dbo:sourcePosition is a property
geo:SpatialThing is in the range of dbo:sourcePosition
dbo:mouthPosition is a property
geo:SpatialThing is in the range of dbo:mouthPosition
geo:lat is a property
geo:SpatialThing is in the domain of geo:lat
geo:lat's object is of the datatype float
```

Listing 2: Ontology that describes people and furniture

```
dbr:river_thames a dbo:River;
  dbo:sourcePosition _:b0;
  dbo:mouthPosition _:b2.

_:b0 a dbo:Place;
  geo:lat "51°41'39\"N".

_:b2 a dbo:Place;
  geo:lat "51°29'56\"N".

dbr:river_cam a dbo:River;
  dbo:sourcePosition _:b1;
  dbo:mouthPosition _:b3.

_:b1 a dbo:Place;
  geo:lat "52°20'54\"N".

_:b3 a dbo:Place;
  geo:lat "52°20'54\"N".

dbr:ghent a dbo:City;
  dbo:location [ a dbo:Place;
       geo:lat "51°2'60\"N" ].

dbr:brussels a dbo:City;
  dbo:location [ a dbo:Place;
    geo:lat "50°50'60\"N" ].
```

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

3. Related work

Knowledge graphs can be generated via rules in different languages (see Section 3.1). These graphs can contain inconsistencies, which can be detected by assessing the graphs' quality (see Section 3.2). Once these inconsistencies are detected they are resolved to improve the knowledge graph (see Section 3.3).

A number of research efforts assume that knowledge graphs are materialized via the Resource Description Framework (RDF) [7, 15]. RDF uses a graph-based model with a set of triples as core structure. Each triple consists of a subject, predicate, and object. A set of triples are called an RDF graph.

3.1. Knowledge graph generation

Two approaches prevailed for knowledge graph generation from existing data sources: direct mapping and custom rules. In the former case, the direct mapping defines a simple transformation, providing a basis for defining more complex transformations afterwards [3]. A Direct Mapping of relational data to RDF was recommended by W3C [3]. However, this recommendation only refers to data existing in relational databases and requires defining rules later, e.g., using SPARQL queries [29], to replace the original predefined annotations with custom ones.

In the latter case, knowledge graph generation languages, e.g., the W3C recommended R2RML [8], RML [11], and SPARQL-Generate [23], offer a declarative way to define rules that specify how knowledge graphs are generated from raw data. [R2]RML rules are in RDF, forming themselves a knowledge graph, i.e., the so-called *rules knowledge graph* (see Fig. 3).

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 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. However, consistently annotating the existing data sources with ontology terms is not always straightforward. Inconsistencies are introduced when the rules are defined. Indicatively, the Semantic Publishing Challenge required different solutions to generate knowledge graphs from the CEUR-WS proceedings [14]. It was observed that, despite the fact that these solutions were provided by Semantic Web experts, the data was modeled differently and each solution introduced different inconsistencies.

3.2. Knowledge graph quality assessment

There exists a number of methods to assess the quality of knowledge graphs. On the one hand, there

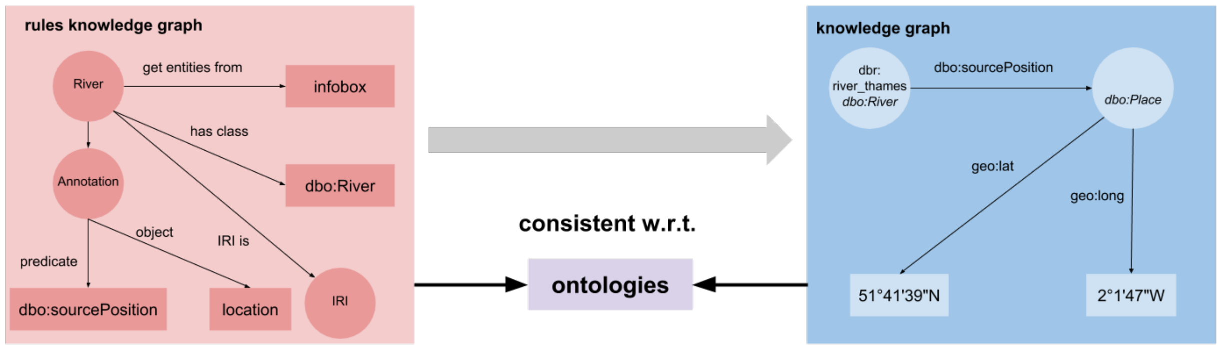


Figure 3. A knowledge graph needs to be consistent with regard to the used ontologies. Therefore, rules, which also form a knowledge graph, also have to be consistent with regard to the same used ontologies.

are methods applied directly to the knowledge graph, based on e.g., crowdsourcing [1], the comparison of the results of queries [16, 22], inference rules [4, 27], evolution analysis [30], or custom characteristics [25]. These methods 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 methods 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 [22]. Such methods 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. [22] 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., does not refer to actual data in the data source. For example, if we have a constraint on the range of a property's numerical value, 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 interval 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.

3.3. 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*.

3.3.1. 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 [25] is a framework for quality assessment and fusion of knowledge graphs. The quality assessment task is realized through a flexible module, where users can choose which characteristics of the data indicate 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 [16] 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 allows seeing all 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.

3.3.2. 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 [28], occurs when the source of knowledge to refine the knowledge graph is not part of the original knowledge graph. In our previous work [12], we proposed such a refinement where the source of the knowledge is the ontologies and the knowledge graph is the rules. Our uniform, iterative, incremental assessment and refinement method for RML rules produces a high-quality knowledge graph, in the form of an RDF dataset (see bottom of Fig. 4). It is created by applying the assessment process, normally applied to the knowledge graph, to the knowledge graph generation rules. This allows discovering inconsistencies, before the knowledge graph is generated. The rules assessment takes significant less time compared to assessing the actual graph which might take a considerable amount of time [13]. Indicatively, assessing the knowledge graph of the English DBpedia takes approximately 16 hours, assessing the knowledge graph of the rules that generate the entire DBpedia takes only 32 seconds. The method consists of the following steps:

1. Inconsistencies are detected via the rules, as it would have been done on the actual dataset.
2. Rules are automatically refined (step 2A at the bottom of Fig. 4) and re-assessed to detect new inconsistencies.
3. The refined version of the rules is 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 the rules. For example, instead of validating the predicate's domain and range against the triple's subject and object respectively, we validate the rules that define how the subject, predicate, and object are generated. The properties and classes in the rules are identified and their schemas are used to generate test cases, as for the actual dataset.

We adjusted the assessment queries by Kontokostas et al. [22] 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, when the literal value of a certain property is constrained within a given range. We refer to the original work [22] which details how the violation patterns are aligned to the rules that should be refined.

Although we introduced a rule-driven inconsistency resolution method, the aforementioned two steps focus on [R2]RML [8, 11], while a similar method 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 [27]. User intervention can be considered to decide if the knowledge graph generation rules or ontologies need to be refined. However, the method does not provide a way to guide users regarding which rules should be inspected first and how these rules and ontologies can be refined.

3.3.3. Ontology-level

Paulheim [27] performed a *data-driven analysis* on the DBpedia rules and concluded that refinements might not only be needed on rules, but also on the ontology definitions. An overview of the steps followed during his analysis can be found here and on the top of Fig. 4:

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 ontology definitions. All rules are marked that contribute to a specific inconsistency. For each rule, two counters are kept: how often the rule generates an assertion involved in an inconsistency (i_m, were m is a rule) 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 calcu-

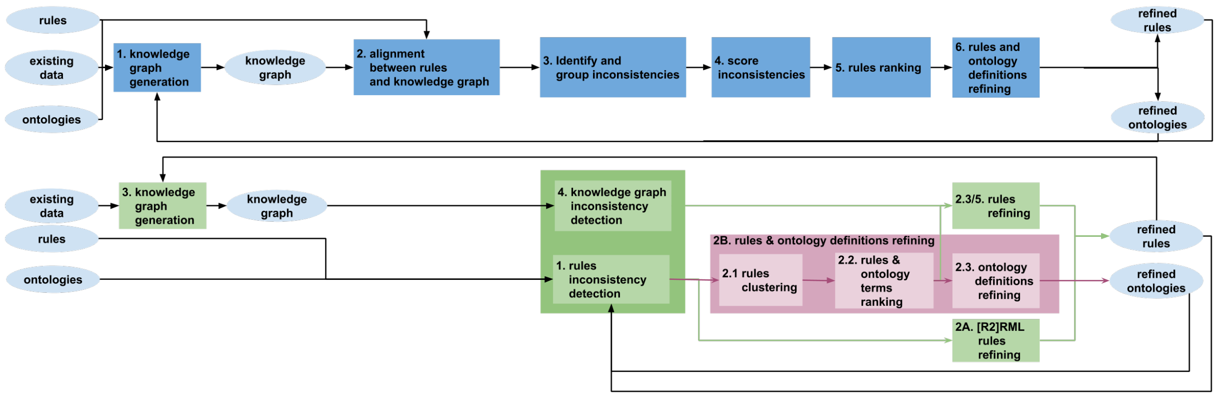

rules
existing data
ontologies
1. knowledge graph generation
knowledge graph
2. alignment between rules and knowledge graph
3. Identify and group inconsistencies
4. score inconsistencies
5. rules ranking
6. rules and ontology definitions refining
refined rules
refined ontologies
existing data
rules
ontologies
3. knowledge graph generation
knowledge graph
4. knowledge graph inconsistency detection
1. rules inconsistency detection
2B. rules & ontology definitions refining
2.1 rules clustering
2.2. rules & ontology terms ranking
2.3. ontology definitions refining
2.3/5. rules refining
2A. [R2]RML rules refining
refined rules
refined ontologies

Figure 4. The top describes the steps (blue rectangles) followed during the analysis of DBpedia by Paulheim [27]. The bottom describes the steps of our previous work [12] (green rectangles) and Resglass (purple rectangles), where steps 1, 3, 4 and 5 from our previous work are reused by Resglass.

Table 1

Comparison of the required input, where they consider refinements, and for what rankings are provided of the different methods. "(x)" is used when the input is not required at the start.

	input				**refinements in**		**ranking of**	
	existing data	**knowledge graph**	**rules**	**ontologies**	**rules**	**ontologies**	**rules**	**ontologies**
Dimou et al. [12]	(x)	(x)	x	x		x		
Paulheim [27]	x	x	x	x	x	x	x	
Resglass	(x)	(x)	x	x	x	x	x	x

lated as $logs(m) = \frac{log(i_m+1)}{log(N+1)}$ and the confidence as $c(m) = \frac{i_m}{i_m+c_m}$, were N is total number of statements in the knowledge graph. 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 definitions.

These steps fill a gap in our previous work (see Table 1), because it provides a set of top ranked rules that should be manually inspected by an expert, similar to Sieve and CLAMS. However, (i) these steps were followed to provide a *preliminary analysis*, but do not form a systematic method that improves our original work on inconsistencies resolution [12]. For instance, the knowledge graph and its generation rules alignment are case specific, namely it is custom to the DBpedia knowledge graph, thus, it can not be applied beyond the scope of the explored use case. (ii) The alignment among the knowledge graphs and rules that generated them needs to be determined separately in a dedicated module, as the provenance of the knowledge graph needs to be reconstructed, but this might not always be accurate or even possible; (iii) the steps are limited to ranking rules and do not support ontology terms ranking, but only refinements to ontology definitions; and (iv) the complete knowledge graph is needed which increases the execution time, making this method unsuitable for use cases that deal with time constraints [13].

Rashid et al. [30] provide a method on evolving knowledge bases. His method looked into identifying completeness issues using knowledge based evolution analysis, and consistency issues based on integrity constraints. However, their method did not look into addressing the results of quality assessment and, thus, do not provide any suggestions with regard to the inconsistencies resolution.

4. Resglass

We propose a new method called Resglass 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 [27] (see Section 3.3.2), and by ranking the ontology terms (see Table 1).

Resglass (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 Resglass can be found here and at the bottom of Fig. 4:

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 the rules and ontology terms.
    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.

4.1. *Rules inconsistency detection*

We validate the rules to determine which inconsistencies are present with respect to used ontologies (see step 1 at the bottom of Fig. 4). 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 [22] (see Section 3.2). This substep does not depend on how the constraints are described, other constraints can also be used, such as SHACL constraints [21].
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 13 in the ontology (see Listing 2). The corresponding instantiated constraint is as follows: for every rule that annotates an entity with the property `geo:lat`, there should exist a rule that annotates that same entity with the class `geo:SpatialThing`. We determine all subsets of rules that could lead to the failure of this constraint. In our example we have two subsets: rules 38 and 43, and 52 and 57 (see Listing 1). The constraint fails because the entity is annotated only with `dbo:Place` and not `geo:SpatialThing`.

4.2. *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 where we use an automatic approach that applies a set of predefined refinements to the rules (see Section 3.3.2, steps 2A and 2B at the bottom of Fig. 4).

4.2.1. *Rules clustering*

The involved rules are clustered based on their contribution to generate a knowledge graph from the different records, e.g., rows in a table or infoboxes in

Wikipedia, in the same data source: the specific type of the record to which a rule contributes is identified, e.g., Wikipedia infoboxes that follow the river template are a single type of record; followed by a grouping of the rules per type of record, e.g., all rules that contribute to the infoboxes of the river template are in the same group. Clustering has been applied in other research domain, such as information exploration [18]. For example, Web search results are clustered, based on a set of similarities, to allow users to quickly browse through the returned documents, because they are related to each other [19, 32]. We applied the same approach to rules where the similarity is determined by the type of record. More specific, rules that contribute to the same type of record are related to each other, i.e., other rules might affect the refinement that needs to be applied to a rule. Therefore, we cluster the rules based on the type of record, so that they can be inspected together.

Example Rules 21, 29, 38, 43, 52, and 57 are involved in inconsistencies (see Listing 1). Rules 38 and 43 are related to the source locations. Rules 52 and 57 are related to the mouth locations. Rules 21 and 29 are related to the river entities. This results in three clusters: one with rules 38 and 43, one with rules 52, and 57 and one with rules 21 and 29.

4.2.2. Rules and ontology term ranking

Once the rules are clustered, we rank both the rules and ontology terms to determine the 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 3.3.3).

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

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

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 three clusters and the five ontology terms. For a cluster, we count the unique inconsistencies in which its rules are involved. The first cluster contains the rules 38 and 43, and the second cluster rules 52 and 57. Both clusters are involved in one inconsistency, resulting in a $score_c$ of 0.25 for both, as there are 4 inconsistencies in total. The third cluster contains the rules 21 and 29. Therefore, the number of inconsistencies is 2, resulting in a $score_c$ of 0.50. This means that the third cluster will be ranked before the other two. Thus, experts will first inspect rules 21 and 29, because they are together involved in the most inconsistencies and are related to the same entity. The five ontology terms are `dbo:Place`, `dbo:sourcePosition`, `dbo:mouthPosition`, `geo:lat`, and `geo:SpatialThing` (see lines 4, 5, 8, 10, and 12 in Listing 2). `dbo:sourcePosition` and `dbo:mouthPosition` are only involved in one inconsistency. Therefore, their corresponding $score_t$ is 0.20. `geo:lat` is involved in two inconsistencies. Therefore, its corresponding $score_t$ is 0.40. `geo:SpatialThing` and `dbo:Place` are involved in all inconsistencies. Therefore, their corresponding $score_t$ is 1. This means that `dbo:Place` and `geo:SpatialThing` will be ranked first, followed by `geo:lat`. Note that clusters and terms can be ranked in different ways when they have the same score.

4.2.3. 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 Resglass 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 Resglass.

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 21 and 29 and observe that the link between a river and its location of the source and mouth are correctly annotated using `dbo:sourcePosition` and `dbo:mouthPosition`. Thus, we leave the rules unchanged. We inspect `dbo:Place` and its corresponding definitions 4 and 7 in the ontology. They state that place is a class and in the range of `dbo:location`. This is still correct and, thus, we leave the definition unchanged. We inspect also `geo:SpatialThing` and its corresponding definitions 5, 9, and 11, 13 in the ontology. We see that `dbo:SpatialThing` is in the domain of `dbo:sourcePosition` and `dbo:mouthPosition`, but the locations are annotated with the class `dbo:Place`. One way to resolve this issue is to remove the rules that annotes a location with the class `dbo:Place` add a new rule that annotates it with `geo:SpatialThing`.

Once the refinements are applied, we restart Resglass. We notice that a new inconsistency was introduced, because we did remove the class `dbo:Place` for a location, while this is class in the range of `dbo:location`. This can be resolved by applying another iteration of Resglass' steps, resulting in, e.g., the annotation of entity with both classes `dbo:Place` and `geo:SpatialThing` or creation of a ontology definition that states that `geo:SpatialThing` is a subclass of `dbo:Place`.

4.3. Knowledge graph generation

The knowledge graph is generated by applying the semantic annotations to existing data sources via rules (see step 3 at the bottom of Fig. 4). This graph does not contain the inconsistencies resolved in the previous steps, because the refined rules and ontology definitions are used.

```
1  dbr:river_thames a dbo:River;
2    dbo:sourcePosition _:b0;
3    dbo:mouthPosition _:b2.
4
5  _:b0 a dbo:Place, geo:SpatialThing;
6    geo:lat "51°41'39\"N".
7
8  _:b2 a dbo:Place, geo:SpatialThing;
9    geo:lat "51°29'56\"N".
10
11 dbr:river_cam a dbo:River;
12   dbo:sourcePosition _:b1;
13   dbo:mouthPosition _:b3.
14
15 _:b1 a dbo:Place, geo:SpatialThing;
16   geo:lat "52°20'54\"N".
17
18 _:b3 a dbo:Place, geo:SpatialThing;
19   geo:lat "52°20'54\"N".
20
21 dbr:ghent a dbo:City;
22   dbo:location [ a dbo:Place;
23     geo:lat "51°2'60\"N" ].
24
25 dbr:brussels a dbo:City;
26   dbo:location [ a dbo:Place;
27     geo:lat "50°50'60\"N" ].
```

Listing 4: Knowledge graph generated by applying the refined rules on the data in Figs. 1 and 2

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

4.4. Knowledge graph inconsistency detection

The knowledge graph is validated to determine inconsistencies (see step 4 at the bottom of Fig. 4). 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 3). Inconsistencies that could not be detected using the rules can be detected in this step.

Example Triples 6, 9, 16, 19, 23, and 27 in Listing 4 state the latitude of the locations and cause six inconsistencies. The ontology definitions require latitude to be given as a float (see line 14 in Listing 2), but this is not the case, leading to a inconsistency every time `geo:lat` is used.

4.5. Rules and ontology definitions refinement

Inconsistencies detected in the previous steps might be resolved by refining the rules and ontology definitions (see step 5B at the bottom Fig. 4). This is different from the last step of our previous work, where only the rules are refined (see step 5A in Fig. 4).

Example Rules can be added to transform the latitudes of locations for use with `geo:lat` instead of the original, unchanged values.

5. Implementation

In this work, we use the RDF Mapping Language (RML) [11] as the underlying knowledge graph generation language to apply Resglass (see Section 5.1), because it is (i) an extension of R2RML [8], the only W3C recommended knowledge graph generation language (see Section 3.1); (ii) used in our previous work [12] (see Section 3.3.2); and (iii) used during the data-driven analysis of DBpedia [27] (see Section 3.3.3). We describe the implementation of the Resglass' steps (see Section 5.2). Note that we only discuss the first four steps, because the others are analogue to the method of our previous work, and not the ontology definitions, because these are independent of the rules and, thus, the language. The complete implementation is available at https://github.com/RMLio/rule-driven-resolution.

5.1. 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 data in CSV, JSON and XML format, as well as files, Web APIs, and so on. RML is extensible, i.e., to data sources in other formats. We describe here the details of the language that are relevant for this work. For the full specification, we refer to http://rml.io/spec.html. A subset of the corresponding RML rules for our motivating use case is given in Listing 1.

For every entity there is a corresponding Triples Map (`rr:TriplesMap`): `<#TriplesMap1>` for the rivers and `<#TriplesMap2>` for the source locations. 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 for the RDF triples' subjects via a template (`rr:template`). 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 three Predicate Object Maps: class, link to the source location, and link to the mouth location. For the Predicate Object Map that defines the class, we use the `rdf:type` as the predicate (`rr:predicate`). Note that the class in this example does not depend on the data. Therefore, it is constant (`rr:constant` in the Object Map). For the Predicate Object Maps that annotate an entity with an attribute, such as the latitude of a location, we use values from the data (`rml:reference` in the Object Map) instead of a constant.

5.2. Resglass

In this section, we discuss implementation of the different steps of Resglass, including accompanying examples. For this we rely on RML as our knowledge graph generation language.

1. Rules inconsistency detection The inconsistencies in RML rules are detected via a rule-based reasoning system [4]. For each constraint type the corresponding inference rules are created. The RML rules, ontologies, and inference rules, which assess the constrains, 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 [4], instead of an approach where the results of queries are compared (see Section 3.2), which we used in our previous work, for: (i) supporting [R2]RML shortcuts via a custom entailment regime, (ii) finding implicit inconsistencies, and (iii) determining the root cause. [R2]RML defines many shortcuts to make it easier for humans to write the rules. This means that different rule sets can generate the same RDF dataset [8]. Thus, every shortcut needs to be defined separately per constraint type in a queries execution approach.

By enabling a custom entailment regime via rule-based reasoning, we can (i) define [R2]RML shortcuts as custom concrete entailment regimes that can be reused for different constraint types; (ii) detect implicit inconsistencies if needed by including another entailment regime [6]; and (iii) precisely determine the root causes of individual inconsistencies using the formal proof, even when including custom entailment regimes, due to the formal logical framework of this reasoning system. When using an approach where the results of queries are compared, we need a different system to reason over custom [R2]RML entailment regimes. The connection with the original rules set is lost and the original root cause cannot be found. However, accurately and correctly identifying root causes across different rule sets important is for Resglass.

example Consider the axiom on line 9 in the ontology (Listing 2). The corresponding instantiated constraint is: for every combination of Term Maps (Predicate Object Map, Predicate Map, and Object Map) that annotates an entity with `dbo:sourcePosition`, the object should refer to an entity with the class `geo:SpatialThing`. Next, we determine all groups of Term Maps that could lead to the failure of this constraint. In our example we have one group consisting of `<#PM1>` and `<#OM2>`. The constraint fails as the location entity is annotated with the class `dbo:Place`.

2. RML rules clustering The RML rules are clustered by determining the Triples Map to which the rules, i.e., Term Maps, correspond. This occurs because 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 determine the corresponding Triples Map via the `rr:predicateObjectMap` that defines the relationship between the former and latter. If the rule is a Subject Map, we determine the corresponding Triples Map via the `rr:subjectMap`. If the rule is a Predicate or Object Map, we determine the Triples Map by first determining the corresponding Predicate Object Map via the `rr:predicateMap` and `rr:objectMap`, respectively. The corresponding Triples Map of the identified Predicate Object Map is determined as described earlier. If the rule is a Triples Map, the corresponding Triples Map is itself.

example If we determine the clusters of the Term Maps `<#PM1>`, `<#PM2>`, `<#PM3>`, `<#PM4>`, `<#OM2>`, and `<#OM3>`, then we have three clusters. The first corresponds with the Triples Map `<#TriplesMap1>`, and contains `<#PM1>` and `<#PM4>`. The second corresponds with the `<#TriplesMap2>`, and contains `<#OM2>` and `<#PM2>`. The third corresponds with the `<#TriplesMap3>`, and contains `<#OM3>` and `<#PM3>`.

3. 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, 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)). The ontology terms ranking does not depend on the used language. Thus, it is done as described in Section 4.2.2.

example The cluster of `<#TriplesMap1>` is involved in two inconsistencies, and `<#TriplesMap2>` and `<#TriplesMap3>` both in one. Thus, the scores of these cluster are 0.50, 0.25, and 0.25, respectively, analogues to our example in Section 4.2.2.

4. 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 4.2.3.

example Let's assume that we only inspect the cluster of `<#TriplesMap2>`. We inspect the Predicate Maps `<#OM2>` and `<#PM2>`. One possibility is to replace `<#OM2>` with a new Object Map that annotates a location with the class `geo:SpatialThing`. Once the refinements are applied, we restart Resglass to determine whether all inconsistencies are resolved or if new inconsistencies were introduced.

5. 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).

6. 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 Triples 6, 9, 16, 19, 23, and 27 in Listing 4 state the latitudes and cause 6 inconsistencies. The ontology definitions require the latitude to be float (see line 14 in Listing 2), but this is not the case, leading to an inconsistency.

7. 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 latitude of the locations instead of the original, unchanged values. This is done by using a function that returns the float version of a latitude. We use the Function ontology [9] to declarative describe this transformation in the RML rules [10]. For the latitude in `<#TriplesMap2>`, we replace rule 41 with rr:objectMap <#parseLatitude>, and we add the following rules:

```
<#parseLatitude>
  fnml:functionValue [
    rr:predicateObjectMap [
      rr:predicate fno:executes;
      rr:objectMap [
        rr:constant ex:parseLatitude]];

    rr:predicateObjectMap [
      rr:predicate ex:inputString;
      rr:objectMap [
        rr:reference "latitude"]]].
```

The data of "latitude" is used as input for the function ex:parseLatitude[3], which returns the float version of a string. For a detailed description about functions, we refer to https://w3id.org/function/.

6. Evaluation

We conducted a comparison to validate our hypothesis (see Section 1). We compare the ranking of the rules and ontology terms provided by experts to the automated ranking provided by Resglass.

In Section 6.1, we elaborate on the evaluation method, namely the procedure and participating experts. In Section 6.2, we discuss the results.

6.1. Method

In this section, we discuss the procedure followed during our evaluation, together with the experts that participated.

Procedure Experts on creation of ontologies and knowledge graph generation languages were directly contacted by the authors. Those who agreed, partook in the following experiment: We selected 25 Wikipedia infobox templates that are annotated with RML rules to generate triples that are part of the DBpedia knowledge graph. The templates selection was random but we opted for the ones whose rules and ontology terms were involved in at least one inconsistency.

Two lists were presented to the experts: (i) a list with URLs of the Triples Maps that correspond with the templates, and (ii) a list with URLs of the ontology terms that are involved in at least one inconsistency. We provided a file for each template containing the inconsistencies in which the specific template is involved. The file includes the type of inconsistency and rules involved, together with the number of rules that might be affected when the involved rule is updated.

The experts had two tasks: to rank the Triples Maps and ontology terms in the order that they would inspect them. The ranking was done by assigned a score to every item in the two lists. The score is a number between 0 and 1 and multiple items can have the same scores. When two items have the same score, both items are equally important to be inspected. The full, detailed instructions and files given to the experts can be found at https://doi.org/10.6084/m9.figshare.7410479.v2.

The random rankings were automatically generated by assigning an integer, representing the rank, to each Triples Map and ontology term. The integers were randomly generated within the range of 1 and the total number of Triples Maps or ontology terms. We generated 100 rankings for the Triples Maps and 100 for the ontology terms. The corresponding code can be found at https://github.com/RMLio/rule-driven-resolution/tree/cluster/resglass-random.

Experts Three experts partook in the experiment, their age range was 28 to 32. All were highly educated: two had a PhD and one a master's degree. They were experts in knowledge graph generation rules and had experience in the use and definition of ontologies.

6.2. Results

We compare the two rankings of each expert with (i) our automatic rankings produced with Resglass and with (ii) random rankings. The comparison is performed using the Ranked-Bias Overlap (RBO) [31]. It is a measure to compare two lists and returns a value between 0 and 1. The measure allows ties of items and puts more weight on the top-ranked items in a list, which are not simultaneously supported by measures like Kendall's tau [20] and Spearman's rho. Ties are possible in case two or more rules, i.e., Triples Maps, or ontology terms are equally important to inspect. More weight is put on the top-ranked items. This aligns with cases where experts fix a subset of the inconsistencies, and then analyze the refined rules and ontology definitions again to find the remaining, and possibly, new inconsistencies.

The average overlap is 81% for the rules and 79% for the ontology terms. The former means that there is a large overlap between the experts' manual ranking

[3]ex is the prefix for the namespace http://example.com/

Table 2
Comparison of the experts', Resglass, and random rankings showing that Resglass has 80% overlap with the experts' and that it improves random rankings by a least 20%.

expert	**RBO (%)**			
	rules		**ontology term**	
	Resglass	random	Resglass	random
1	94	34	78	56
2	59	45	81	57
3	90	37	78	56
average	81	39	79	56

and our automatic ranking when more attention is put to the top ranked rules then the lower ranked rules. The latter means that again there is a large overlap between the experts' ranking and our automatic ranking.

Furthermore, our automatic ranking improves the overlap with experts' manual ranking with 41% for rules and 23% for ontology terms, compared to a random ranking. The RBO values for each expert's rules and ontology term rankings can be found in Table 2.

7. Conclusion

Knowledge graphs are often generated by applying generation rules that annotate data in existing data sources with ontology 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 method called Resglass that ranks the rules, via clustering, and ontology terms involved in an inconsistency. The top rules and ontology terms are inspected by experts and the necessary refinements are applied to resolve the inconsistencies.

Refinements can be applied to both rules and ontology terms 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? Resglass, 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.

Our evaluation with experts shows that our ranking has 80% overlap with the experts' rankings for both the rules and the ontology elements. Furthermore, our ranking improves the overlap with experts' with 41% for rules and 23% for ontology terms, compared to a random ranking. Thus, this evaluation provides evidence towards the acceptance of our hypothesis.

The ranking proposed of Resglass 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. Special thanks to Tom Seymoens and Io Taxidou for their valuable advice.

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.

[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.

[3] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. A Direct Mapping of Relational Data to RDF. Working Group Recommendation, W3C, 2012. URL http://www.w3.org/TR/rdb-direct-mapping/.

[4] 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, 2017. .

[5] 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. .

[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, 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 Steinmetx, 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, 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. .

[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, 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] Anastasia Dimou, Sahar Vahdati, Angelo Di Iorio, Christoph Lange, Ruben Verborgh, and Erik Mannens. Challenges as enablers for high quality Linked Data: insights from the Semantic Publishing Challenge. *PeerJ Computer Science*, 3:e105, January 2017. ISSN 2376-5992. . URL https://doi.org/10.7717/peerj-cs.105.

[15] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked Data Quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. *Semantic Web*, pages 1–53, 2016.

[16] 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. .

[17] Nicola Guarino. Ontologies and knowledge bases: towards a terminological clarification, 1995.

[18] Marti A Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4): 59–61, 2006.

[19] Mika Käki and Anne Aula. Findex: improving search result use through automatic filtering categories. *Interacting with Computers*, 17(2):187–206, 2005.

[20] Maurice G Kendall. Rank Correlation Methods. 1955.

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

[22] 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. .

[23] 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.

[24] 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.

[25] 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. .

[26] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language. Technical report, W3C, 2012. URL https://www.w3.org/TR/owl2-syntax/.

[27] 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.

[28] Heiko Paulheim. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic web*, 8(3):489–508, 2017.

[29] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. Technical report. URL http://www.w3.org/TR/rdf-sparql-query/.

[30] Mohammad Rashid, Giuseppe Rizzo, Marco Torchiano, Nandana Mihindukulasooriya, Oscar Corcho, and Raúl GarcíaCastro. Completeness and consistency analysis for evolving knowledge bases. *Journal of Web Semantics*, 2018. .

[31] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems*, 28(4):20, 2010.

[32] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma. Learning to cluster web search results. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 210–217. ACM, 2004.