Enabling Completeness-aware Querying in SPARQL

Luis Galárraga
Aalborg University
galarraga@cs.aau.dk

Katja Hose
Aalborg University
khose@cs.aau.dk

Simon Razniewski
Free University of Bozen-Bolzano
razniewski@inf.unibz.it

ABSTRACT

Current RDF knowledge bases (KBs) are highly incomplete. This incompleteness is a serious problem both for data users and producers. Users do not have guarantees that queries that are run on a KB deliver complete results. Data producers, on the other hand, are blind about the parts of the KB that are incomplete. Yet, completeness information management is poorly supported in the Semantic Web. No RDF storage engine supports reasoning with completeness statements. Moreover, SPARQL cannot express completeness constraints for queries. Motivated by these observations, this paper offers a vision on completeness-aware RDF querying. Our vision includes (1) the sketch of a method to reason about completeness in RDF knowledge bases, (2) two approaches to represent completeness information for SPARQL queries, and (3) an extension for the SPARQL language to express completeness constraints in queries.

1 INTRODUCTION

In the past 15 years we have seen a steady increase in the amount of available semantic data on the Web[1]. Semantic data is normally modeled in RDF [12] as facts ⟨*subject, relation, object*⟩. We call a collection of RDF facts a *knowledge base* (KB). Current KBs suffer from quality problems. Those problems include false and missing information. While semantic data providers have traditionally focused on the correctness of the information, the dimension of completeness has only recently attracted attention of the research community [1–3, 7, 8]. Nevertheless, existing KBs are highly incomplete. As of 2015, Wikidata [11], for example, knows the father of only 2% of the people in its records. In [9] the authors show that between 69% and 99% of the entities in popular KBs lack at least one property that other entities in the same class have. Moreover, due to the open-world assumption, it is sometimes impossible to detect where the information is missing: a person without a spouse in the KB may be truly single or married to an unknown person. These scenarios are problematic both for data users and data producers. Data users do not have guarantees about the completeness of query results on KBs. In contrast, data producers cannot know which parts of the KB should be populated.

There have been, though, some efforts to alleviate this problem. At the time of writing, Wikipedia contains more than 2900 lists asserted as incomplete[2]. However, if a list is not asserted as incomplete, it does not necessarily mean that the list is complete. Also, Wikidata [11] defines no-value statements: assertions specifying that an entity does not have values for a certain attribute. Still, these assertions have a limited scope, e.g., they cannot tell us if a person with a known citizenship has more citizenships in real life. The work in [3] studies methods to predict completeness statements in YAGO [10] and Wikidata [11]. Yet, they can only predict completeness for the list of objects of a given subject and relation. Despite these efforts, the dimension of completeness remains significantly unexplored. First, the available completeness assertions are defined for *simple queries*, e.g., lists with simple definitions such as the "list of Nobel laurates". A KB may be incomplete for this list and still complete for the—more complex to describe—list of Nobel laurates in Physics. Second, we cannot use those completeness assertions to provide completeness guarantees for arbitrary queries, because no RDF storage engine nowadays supports inference with completeness statements. If a KB contains completeness statements about the list of Nobel Prize laurates of each category, then it follows that the KB is complete for the entire set of Nobel laurates. Nowadays, such a reasoning is not possible on semantic data—albeit formalized in [1, 2, 6]. Third, SPARQL does not provide a way to express completeness constraints on RDF data, i.e., it is not possible to write queries for regions of the data asserted as complete. Thus, in this paper we present a vision of a Semantic Web that is aware of its incompleteness. Our vision consists of three points: (1) a vision for reasoning with completeness statements, (2) two representations for completeness statements in RDF data, and (3) a proposal to extend the SPARQL query language in order to support completeness constraints.

The remainder of this paper is structured as follows. In Sec. 2, we describe the basic concepts that are relevant to our vision of completeness. Sec. 3 elaborates on our ideas on reasoning with completeness. Sec. 4 describes our proposals for the representation of completeness statements for RDF. Sec. 5 describes how to extend SPARQL to support completeness constraints. Finally, Sec. 6 concludes the paper.

2 PRELIMINARIES

2.1 RDF Knowledge Bases and SPARQL

We assume that the reader is familiar with RDF [12] and SPARQL [13]. An RDF *knowledge base* (KB) is a collection of facts in the form of triples $\langle s, r, o \rangle$ where s is the subject, r is the relation, and o is the object, e.g., ⟨*Denmark, capital, Copenhagen*⟩. SPARQL [13] is designated by the W3C as standard for querying RDF data. In this paper we focus on a subset of the standard, namely the set of SPARQL conjunctive queries with aggregates. For space reasons, we do not provide a rigorous definition of SPARQL queries; instead we refer to the definition used in [5].

2.2 Completeness in KBs

In line with previous work on completeness in RDF data [1, 3], we define the completeness of a KB with respect to queries. We say a KB $\mathcal{K}$ is complete w.r.t. a query q, if q delivers at least the same results in $\mathcal{K}$ as in $\mathcal{K}^*$, i.e., $q(\mathcal{K}) \supseteq q(\mathcal{K}^*)$. Here, $\mathcal{K}^*$ denotes a hypothetical complete KB that knows all the results of q that hold in

[1] https://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets offers an extensive list of publicly available datasets.
[2] https://is.gd/j2eb9f

subject	relation	object
Barack Obama	rdf:type	Politician
Barack Obama	citizenOf	USA
Angelina Jolie	citizenOf	USA

Table 1: Example KB

the real world. For example, given the SPARQL query q "SELECT *?country* WHERE { *Spanish officialLangIn ?country*}", we say a KB $\mathcal{K}$ is complete w.r.t this query if $\mathcal{K}$ knows all the countries in the world where Spanish is an official language.

2.3 Completeness oracles

Based on the work presented in [3], we define a *completeness oracle* $\omega(q, \mathcal{K})$ as a boolean function. The function returns true whenever the oracle believes that the query q is complete in $\mathcal{K}$. In this work, we do not distinguish between "incomplete" or "unknown completeness status". We omit $\mathcal{K}$ from the arguments whenever it is clear that we are talking about a single KB. In [3] the authors propose and study a set of completeness oracles for queries of the form q: SELECT *?o* WHERE { *s r ?o* }. These oracles predict whether a KB knows all the object values of a given subject-relation pair $\langle s, r\rangle$. We call them *subject-relation oracles*. For simplicity, we rewrite $\omega(q)$ as $\omega(s, r)$. As an example, consider the KB $\mathcal{K}$ depicted in Table 1 and the following subject-relation oracles:

$$pca(s, r) : \exists\, o : \langle s, r, o\rangle \in \mathcal{K} \qquad pol(s, r) : \langle s, is, Politician\rangle \in \mathcal{K}$$

The *pca* oracle implements the Partial Completeness Assumption (PCA) [4], which states that a pair $\langle s, r\rangle$ is complete if the KB knows at least one object value for the pair. From this definition it follows that *pca*(*B. Obama*, *citizenOf*) and *pca*(*A. Jolie*, *citizenOf*) evaluate to true. The *pol* oracle states that entities in the class of politicians are always complete in their attributes. This implies that *pol*(*B. Obama*, *citizenOf*) evaluates to true, whereas *pol*(*A. Jolie*, *citizenOf*) evaluates to false. We highlight that those oracles can make mistakes, e.g., *pca* is wrong for Angelina Jolie since the KB misses the fact that Angelina is also a citizen of Cambodia. If Ω is the golden oracle that knows the completeness of all subject-relation queries, precision and recall for a subject-relation oracle ω w.r.t a relation r are defined as follows:

$$precision_r(\omega) = \frac{hits_r}{\#s : \omega(s, r)} \qquad recall_r(\omega) = \frac{hits_r}{\#s : \Omega(s, r)} \tag{1}$$

Here $hits_r = \#s : \omega(s, r) \wedge \Omega(s, r)$ is the number of entities for which ω predicted completeness correctly. For Table 1, *pca* has precision 0.5 and recall 1 for the relation *citizenOf*, since *pca* errs for Angelina Jolie but is correct for all the actual complete entities in the KB (Obama). Both precision and recall for *pol* are equal to 1.

A *domain oracle* is a completeness oracle that asserts the completeness of queries of the form: SELECT DISTINCT (*?s* |*?o*) WHERE {*?s r ?o*}. That is, the oracle evaluates to true if the KB knows all the subjects (or objects) that occur with a given relation r in the real world. We denote domain oracles by ω_s and ω_o depending on whether the projection variable is the subject or the object of the triple pattern. For example, if $\omega_o(isCitizenOf)$ evaluates to true, then according to ω_o the KB knows at least one *citizenOf* fact for every country in the world. For simplicity, we write $\omega_o(r)$ as $\omega_s(r^{-1})$, where r^{-1} is r's inverse, i.e., the relation obtained by swapping the arguments of r in the KB, e.g., $isCitizenOf^{-1} = hasCitizen$.

It is easy to see that domain completeness does not entail subject-relation completeness. If a KB knows all the subjects for the relation *isCitizenOf* (all people in the world), it may still miss one of the nationalities of a particular person. For this reason, these two types of oracles can be used as complementary building blocks to infer completeness for more complex queries, as we show next.

3 REASONING WITH COMPLETENESS ORACLES

In our vision, subject-relation and domain oracles are used to construct composite oracles that can provide completeness guarantees for arbitrary SPARQL conjunctive queries under bag semantics.

3.1 Composite Oracles

Consider the completeness oracles ω, ω_s, and the query q': SELECT *?cnt* WHERE { *?cnt officialLang ?l . ?l family Romance*}. In the absence of a completeness statement tailored for q', we can use completeness oracles to infer the completeness of q' by defining a composite oracle ω' as follows:

$$\omega' = \omega(Romance, family^{-1}) \wedge \bigwedge_{l:family(l, Romance)} \omega(l, officialLang^{-1})$$

In other words, ω' will mark q' as complete, if (1) the KB is complete in the list of languages of the Romance family, and (2) for each of those languages the KB is complete in the list of countries where the language has official status. Now imagine somebody devised an oracle ω_h that trivially returns true for this query, based on manual checking of the result of q'. If the list of romance languages in the KB misses Ligurian, ω' will return false for q' even though this particular query is complete. To see this, recall that Ligurian is not official in any country. This scenario tells us that ω' may have too many requirements to evaluate to true, and is therefore not *tight*. In the following, we formalize this notion for completeness oracles.

Definition 3.1. **Tightness of completeness oracles.** Given two completeness oracles ω_1 and ω_2, and a query q, we say that ω_1 is *tighter* than ω_2 for q (denoted as $\omega_1 \prec_q \omega_2$) if

$$\forall\, \mathcal{K} : \omega_1(q, \mathcal{K}) \wedge \omega_2(q, \mathcal{K}) : \exists\, \mathcal{K}' \subset \mathcal{K} : \omega_1(q, \mathcal{K}') \wedge \neg\omega_2(q, \mathcal{K}')$$

According to this definition, given two completeness oracles ω_1 and ω_2 and a query q, we say that ω_1 is tighter than ω_2 if for each KB $\mathcal{K}$ where both oracles evaluate to true, ω_1 can still evaluate to true in a less populated version of $\mathcal{K}$, which we denote by $\mathcal{K}'$.

3.2 Automatic Oracle Composition

In the following we sketch an algorithm that can infer completeness for SPARQL conjunctive queries under bag semantics. The algorithm is based on compositions of subject-relation and domain oracles. We describe the algorithm in a bottom-up fashion by first defining composite oracles for simple queries with one projection variable. Those oracles are used as building blocks to construct completeness oracles for arbitrary SPARQL conjunctive queries with one projection variable. We also discuss briefly the applicability of the method for queries with aggregations and queries under set semantics. We leave as future work the support for multiple projection variables.

Consider the subject-relation and domain oracles ω, ω_s defined for a KB $\mathcal{K}$, and a SPARQL query q of the form SELECT *?v* WHERE { G_p }, where G_p is a basic graph pattern [5]. We denote by $complete(q, \langle\omega, \omega_s\rangle, \mathcal{K})$ the composite oracle constructed by our method. We observe that for queries with a single *selective triple pattern* of the form $t = \langle ?v, r, C\rangle$ or $t = \langle C, r, ?v\rangle$ (C is a constant value), $complete(q, \langle\omega, \omega_s\rangle, \mathcal{K})$ evaluates to either $\omega(C, r^{-1})$ or $\omega(C, r)$. In the following we describe how to implement $complete(q, \langle\omega, \omega_s\rangle, \mathcal{K})$ for different types of simple SPARQL conjunctive queries.

Selective star patterns. A selective star pattern S consists of a set of selective triple patterns. The completeness of a query consisting of a single selective star pattern can be evaluated as

$$complete(q, \langle\omega, \omega_s\rangle, \mathcal{K}) = \bigwedge\nolimits_{t \in S} complete(q_t, \langle\omega, \omega_s\rangle, \mathcal{K})$$

Here, q_t is a query that contains only the selective triple pattern t.

Non-selective triple patterns. If a query q with selection variable $?v$ contains a single non-selective triple pattern of the form $t = \langle ?v, r, ?v'\rangle$ or $t = \langle ?v', r, ?v\rangle$, the completeness of q for the first case can be assessed with the expression:

$$complete(q, \langle\omega, \omega_s\rangle, \mathcal{K}) = \omega_s(r) \wedge \left(\bigwedge\nolimits_s \omega(s, r)\right)$$

The second case can be addressed by replacing r with r^{-1}.

Subgraph patterns. We define a subgraph pattern p as a set of triple patterns containing a single non-selective triple pattern t on the projection variable $?v$, which we call the head, and a transitively connected set of triple patterns not containing $?v$, known as the tail. One example is p ={⟨*?v, citizenOf, ?country*⟩, ⟨*?country, hasCity, ?city*⟩, ⟨*?city, timeZone, UTC-5*⟩ }. The completeness of a query q_p with projection variable $?v$ consisting of one subgraph pattern p can be evaluated with the following formula:

$$complete(q_p, \langle\omega, \omega_s\rangle, \mathcal{K}) = complete(q_{tail}, \langle\omega, \omega_s\rangle), \mathcal{K}) \wedge \left(\bigwedge\nolimits_{e \in q_{tail}(\mathcal{K})} complete(q_t^e, \langle\omega, \omega_s\rangle, \mathcal{K})\right)$$

In this formula, q_t^e is a version of the query such that it contains only the head triple pattern t, and the head non-projection variable ($?country$ in our example) has been instantiated with value e. On the other hand, q_{tail} denotes our original query on the tail of the subgraph pattern but with $?country$ as projection variable. Subgraph patterns cover queries with at most one path-shaped pattern starting at the projection variable, plus arbitrary patterns on the non-projection variables. Therefore, any connected basic graph pattern can be expressed as a combination of multiple subgraph patterns and one (optional) selective star pattern on the projection variable.

Arbitrary conjunctive queries. Algorithm 1 describes an oracle to answer completeness for conjunctive queries with an arbitrary basic graph pattern and with one selection variable. The algorithm takes as input the query q with projection variable $?v$, a subject-relation oracle ω, a domain oracle ω_s, and a KB $\mathcal{K}$. The algorithm starts by identifying the selective star pattern S containing $?v$ in the query (line 1). If such a pattern exists, the algorithm evaluates its completeness according to the oracles (line 4). If the oracles do not evaluate to *false*, the algorithm continues by identifying all subgraph patterns starting at the projection variable. This is done as follows. First, the algorithm gathers all the non-selective triple patterns (line 6) that contain the projection variable $?v$, i.e., the heads of the subgraph patterns. Then, for each head triple pattern $t = \langle ?v, r, ?v'\rangle$, the algorithm identifies the tail (line 9) and constructs a new select query on the complete subgraph pattern (line 10). The algorithm computes the completeness of the query as the conjunction of the completeness assessments of every subgraph pattern (line 12). For example, given a query with projection variable $?v$ and triple patterns {⟨*?v, profession, scientist*⟩, ⟨*?v, citizenOf, ?country*⟩, ⟨*?country, hasCity, ?city*⟩, ⟨*?city, timeZone, UTC-5*⟩}, Algorithm 1 builds an oracle based on the completeness of the selective star pattern {⟨*?v, profession, scientist*⟩} and the subgraph pattern with head ⟨*?v, citizenOf, ?country*⟩ and tail {⟨*?country, hasCity, ?city*⟩, ⟨*?city, timeZone, UTC-5*⟩}.

Algorithm 1: isComplete

Input: q : *SELECT ?v WHERE* { T }, oracles $\langle\omega, \omega_s\rangle$, KB $\mathcal{K}$
Output: true or false

1 $S := \{t \in T : t = \langle C, r, ?v\rangle \vee t = \langle ?v, r, C\rangle\}$
2 **if** $S \neq \emptyset$ **then**
3     $q_s :=$ *SELECT ?v WHERE* { S }
4   **if** $\neg isComplete(q_s, \langle\omega, \omega_s\rangle, \mathcal{K})$ **then**
5   **return** *false*
6 $N := \{t \in T : t = \langle ?v', r, ?v\rangle \vee t = \langle ?v, r, ?v'\rangle\}$
7 $C := \{\}$
8 **for** $t \in N$ **do**
9     $p := \{t' \in T - (S \cup N) : t'$ transitively connected to $t\}$
10     $q_p :=$ *SELECT ?v WHERE* { $p \cup \{t\}$ }
11     $C := C \cup \{q_p\}$
12 **return** $\bigwedge_{q_p \in C} isComplete(q_p, \langle\omega, \omega_s\rangle, \mathcal{K})$

Tightness of our methods. It is easy to see that Algorithm 1 does not produce tight oracles, i.e., the resulting oracles will lead to false negatives in highly incomplete KBs. Given the query that asks for the bag of countries with official romance languages in Section 3.1, Algorithm 1 returns the oracle ω', which can lead to false negatives as we showed. While our method could be applied to queries with set semantics, the produced oracles are even less tight in this case. For example, consider the set-semantics version of our example query and the oracle ω' from Section 3.1. If a KB misses the single fact that Spanish is an official language of Equatorial Guinea, the oracle ω' will return false, even though Equatorial Guinea will appear in the list thanks to French. If a query contains an aggregate term of the form $f(?v)$ in the projection, Algorithm 1 could be applied to the query without the aggregate, however with potential false negatives.

4 REPRESENTING COMPLETENESS STATEMENTS

In this section we propose two conceptual representations for completeness oracles: extensional and intensional.

4.1 Extensional Approach

Under the extensional representation, a completeness oracle is a collection of completeness statements, that is, assertions about the completeness of queries on an RDF KB. In [1], the authors use RDF to model completeness statements for SPARQL conjunctive queries.

```
pcaFuncRelation  rdf:type   SubjectRelationComplOracle .
pcaFuncRelation  formula    "λ ⟨s, r⟩, 𝒦 : ⟨s, r, o⟩ ∈ 𝒦"  .
```

Figure 1: The completeness oracle based on the Partial Completeness Assumption (PCA) described as an RDF resource.

In an extensional representation, an oracle ω could be a named graph with multiple completeness statements. A call to the oracle, e.g., $\omega(q, \mathcal{K})$ will thus trigger a SPARQL ASK query on the named graph associated to ω. This query asks for the existence of a statement for query q in the graph.

4.2 Intensional Approach

The extensional approach can become extremely verbose for subject-relation oracles. Consider the PCA oracle defined in Section 2.3. This oracle has 100% precision for functional relations such as place of birth. These are relations where each subject has at most one object value. Under the extensional approach, each person with known place of birth will produce a completeness statement. In contrast, under an intensional representation, the PCA oracle can be defined as a lambda function on pairs subject-relation as Figure 1 shows. The variable r in the formula could be replaced with the name of any functional relation. A call to a subject-relation oracle $\omega(s, r)$ under the intensional approach triggers a call to the lambda function of the oracle with arguments $\langle s, r \rangle$. The lambda approach for completeness oracles can easily be implemented for some of the subject-relation oracles defined in [3]. In contrast, it is not portable to arbitrary conjunctive queries. Nonetheless, both the intensional and the extensional could be used in combination.

5 COMPLETENESS-AWARE QUERYING

Due to the open-world nature of Semantic Web data, RDF does not provide a native way to handle completeness information. This problem extends directly to the SPARQL query language [13]. Imagine a user who needs to retrieve the total number of Spanish speakers per state in USA by aggregating the number of Spanish speakers in every county of the state. Furthermore, assume the user wants the results for states with complete information, that is, those states where the complete list of counties as well as the total number of Spanish speakers is known. Such a query cannot be expressed in SPARQL. We therefore propose to extend the language to support completeness constraints like the following:

SELECT *?state* sum(*?nspeak*) WHERE {
  ?county inState ?state . *?county spanishSpeakers ?nspeak*
} GROUP BY ?state HAVING (**complete**(*?nspeak*))

In this example *complete*$(?v_1, \ldots ?v_m)$ is a boolean aggregation function applied to each group of counts. For each binding of grouping variables (those in the GROUP BY clause), this construct evaluates completeness on the bindings of $?v_1, \ldots ?v_m$. For example, when the grouping variable *?state* binds to Texas, the evaluation of the aggregation function resorts to a completeness oracle to assess the following intermediate query:

SELECT *complete*(?nspeak) WHERE {
  ?county inState Texas . *?county spanishSpeakers ?nspeak* . }

In this case the *complete* function is applied to the implicit singleton group that contains all the bindings of the variable *nspeak* for counties in Texas. The function returns true whether this set of values is complete according to its underlying oracle. Such an oracle could be generated using Algorithm 1 for example. Finally, RDF query engines could provide a confidence score for completeness answers. This score would depend on the precision of the underlying oracles used to compute the answer.

6 CONCLUSION

We have presented a vision on enabling completeness-aware querying on RDF data. Our vision comprises a framework to reason about completeness based on completeness oracles. This includes a basic method to infer completeness from simple oracles. We also have presented a vision on how to model completeness information for RDF and integrate completeness constraints into the SPARQL language. Our ideas focus on the set of SPARQL conjunctive queries with aggregation. In this paper, we have not discussed other interesting ideas, such as the management of explicit incompleteness information, i.e., in the form of incompleteness oracles, or optimal oracle selection in the presence of multiple oracles with different values of recall. Nevertheless, we believe that our vision is a step forward towards a completeness-aware Semantic Web, and we hope it motivates further research in the area of completeness in RDF.

Acknowledgements This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301, and the Free University of Bozen-Bolzano under the project "TaDaQua".

REFERENCES

[1] Fariz Darari, Werner Nutt, Giuseppe Pirro, and Simon Razniewski. Completeness Statements about RDF Data Sources and their Use for Query Answering. In *Proceedings of the International Semantic Web Conference*, 2013.
[2] Fariz Darari, Simon Razniewski, Radityo Eko Prasojo, and Werner Nutt. Enabling Fine-Grained RDF Data Completeness Assessment. In *Proceedings of the International Conference on Web Engineering*, 2016.
[3] Luis Galárraga, Simon Razniewski, Antoine Amarilli, and Fabian Suchanek. Predicting Completeness in Knowledge Bases. In *Proceedings of the Conference on Web Search and Data Mining*, 2017.
[4] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *The VLDB Journal*, 24(6):707–730, 2015.
[5] Dilshod Ibragimov, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. Optimizing Aggregate SPARQL Queries Using Materialized RDF Views. In *Proceedings of the International Semantic Web Conference*, 2016.
[6] Werner Nutt, Sergey Paramonov, and Ognjen Savkovic. Implementing Query Completeness Reasoning. In *Proceedings of the Conference on Information and Knowledge Management*, 2015.
[7] Radityo Eko Prasojo, Fariz Darari, Simon Razniewski, and Werner Nutt. Managing and consuming completeness information for wikidata using COOL-WD. In *Workshop on Consuming Linked Data*, 2016.
[8] S. Razniewski, F. Korn, W. Nutt, and D. Srivastava. Identifying the Extent of Completeness of Query Answers over Partially Complete Databases. In *Proceedings of the International Conference on Management of Data*, 2015.
[9] F. M. Suchanek, D. Gross-Amblard, and S. Abiteboul. Watermarking for Ontologies. In *Proceedings of the International Semantic Web Conference*, 2011.
[10] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A core of Semantic Knowledge. In *Proceedings of Conference on World Wide Web*, 2007.
[11] D. Vrandečić and M. Krötzsch. Wikidata: A Free Collaborative Knowledge Base. *Communications of the ACM*, 2014.
[12] Word Wide Web Consortium. RDF Primer (W3C Recommendation 2004-02-10). http://www.w3.org/TR/rdf-primer/, 2004.
[13] Word Wide Web Consortium. SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/, 2008.