

The Role of Reasoning for RDF Validation

Thomas Bosch
GESIS - Leibniz Institute for
the Social Sciences, Germany
thomas.bosch@gesis.org

Erman Acar
University of Mannheim,
Germany
erman@informatik.uni-
mannheim.de

Andreas Nolle
Albstadt-Sigmaringen
University, Germany
nolle@hs-albsig.de

Kai Eckert
Stuttgart Media University,
Germany
eckert@hdm-stuttgart.de

ABSTRACT

For data practitioners embracing the world of RDF and Linked Data, the openness and flexibility is a mixed blessing. For them, data validation according to predefined constraints is a much sought-after feature, particularly as this is taken for granted in the XML world. Based on our work in the DCMI RDF Application Profiles Task Group and in cooperation with the W3C Data Shapes Working Group, we published by today 81 types of constraints that are required by various stakeholders for data applications. These constraint types form the basis to investigate the role that reasoning and different semantics play in practical data validation, why reasoning is beneficial for RDF validation, and how to overcome the major shortcomings when validating RDF data by performing reasoning prior to validation. For each constraint type, we examine (1) if reasoning may improve data quality, (2) how efficient in terms of runtime validation is performed with and without reasoning, and (3) if validation results depend on underlying semantics which differs between reasoning and validation. Using these findings, we determine for the most common constraint languages which constraint types they enable to express and give directions for the further development of constraint languages.

Categories and Subject Descriptors

F.4.3 [Mathematical Logic and Formal Languages]: [formal languages]; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods; I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence

Keywords

RDF Validation, RDF Constraint Types, Data Quality, Reasoning, OWL, Linked Data, Semantic Web

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SEMANTICS '15, September 15 - 17, 2015, Vienna, Austria

© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3462-4/15/09...\$15.00

DOI: <http://dx.doi.org/10.1145/2814864.2814867>

1. INTRODUCTION

Recently, RDF validation as a research field gained speed due to common needs of data practitioners. A typical example is the library domain that co-developed and adopted Linked Data principles very early. For libraries, the common description of resources are key business and they have a long tradition in developing and using interoperable data formats. While they embrace the openness of Linked Data and the data modeling principles provided by RDF, the data is still mostly represented in XML and this is unlikely to change soon. Among the reasons for the success of XML is the possibility to formulate fine-grained constraints to be met by the data and to validate the data according to these constraints using powerful systems like DTDs, XML Schemas, RELAX NG, or Schematron. A typical example is the definition of a library record describing a book. There are clear rules which information has to be available to describe a book properly, but also how information like an ISBN number is properly represented. Libraries seek to make their own data reusable for general purposes, but also to enrich and interlink their own data. Checking if third-party data meets own requirements or validating existing data according to new needs for a Linked Data application are among common use cases for RDF validation.

In 2013, the W3C invited experts from industry, government, and academia to the *RDF Validation Workshop*,¹ where first use cases for constraint formulation and RDF data validation have been discussed. Two working groups that follow up on this workshop have been established in 2014 to develop a language to express constraints on RDF data: the *W3C RDF Data Shapes Working Group*² and the *DCMI RDF Application Profiles Task Group*.³ To formulate constraints and to validate RDF data, several languages exist or are currently developed like *Shape Expressions (ShEx)*, *Resource Shapes (ReSh)*, and *Description Set Profiles (DSP)*. The *Web Ontology Language (OWL)* in its current version 2 is also used as a constraint language. With its direct support of validation via SPARQL, the *SPARQL Inferencing Notation (SPIN)* is very popular and certainly plays an important role for future developments of constraint languages. SPIN is particularly interesting as a means to

¹<http://www.w3.org/2012/12/RDF-val/>

²<http://www.w3.org/2014/rds/charter>

³<http://wiki.dublincore.org/index.php/RDF-Application-Profiles>

validate arbitrary constraint languages by mapping them to SPARQL [4]. As there is no clear favorite and none of the languages is able to meet all requirements raised by data practitioners, further research and development is needed.

1.1 Motivation and Overview

Within the DCMi working group, we initiated a collaboratively curated database of RDF validation requirements which contains the findings of the working groups based on various case studies provided by data institutions [3]. The database, which is publicly available and open for further contributions,⁴ connects requirements to use cases, case studies, and solutions. Based on our work in the DCMi and in cooperation with the W3C working group, we published by today 81 requirements to validate RDF data and to formulate RDF constraints that are required by various stakeholders for data applications. Each of these requirements corresponds to a constraint type from which concrete constraints are instantiated to be checked on RDF data. Requirements are uniquely identified in the database by an R and a number. *Minimum qualified cardinality restrictions*, e.g., is a constraint type which corresponds to the requirement *R-75*. We recently published a technical report [5] (serving as appendix of this paper) in which we explain each requirement/constraint type in detail and give examples for each represented by different constraint languages. These constraint types form the basis to investigate the role that reasoning and different semantics play for RDF validation.

Validation and reasoning are closely related. Reasoning is beneficial for RDF validation as (1) it may resolve constraint violations, (2) it may cause valuable violations, and (3) it solves the redundancy problem. A major shortcoming when validating RDF data is *redundancy*. Consider that a publication must have a publication date which is a typical constraint. When defining *books*, *conference proceedings*, and *journal articles* as sub-classes of *publication*, one would require to assign the concerned constraint explicitly to each sub-class, since each of them should have a publication date. Reasoning is a promising solution as pre-validation step to overcome this shortcoming. *Reasoning* in Semantic Web refers to logical reasoning that makes implicitly available knowledge explicitly available. When performing reasoning one can infer that books must have a publication date from the facts that books are publications and publications must have a publications date. We remove redundancy by associating the constraint with the super-class *publication*.

Users should be enabled to select on which constraint types to perform reasoning before data is validated and which constraint types to use to ensure data accuracy and completeness without reasoning. As reasoning is beneficial when validating RDF data, we investigate the effect of reasoning to the validation process of each constraint type, i.e., we examine for each constraint type if reasoning may be performed prior to validation to enhance data quality either by resolving violations or by raising valuable violations (Section 2).

For each constraint type, we investigate how efficient in terms of runtime validation is performed with and without reasoning. By mapping to *Description Logics* (DL) we get an idea of the performance of each constraint type in worst case, since the combination of DL constructs needed to express a constraint type determines its computational com-

plexity (Section 2). For this reason, the appendix of this paper contains mappings to DL to determine which DL constructs are needed to express each constraint type. Thus, the knowledge representation formalism DL, with its well-studied theoretical properties, provides the foundational basis for constraint types.

Validation and reasoning assume different semantics which may lead to different validation results when applied to particular constraint types. Reasoning requires the *open-world assumption* (OWA) with the *non-unique name assumption* (nUNA), whereas validation is classically based on the *closed-world assumption* (CWA) and the *unique name assumption* (UNA). Therefore, we investigate for each constraint type if validation results differ (1) if the CWA or the OWA and (2) if the UNA or the nUNA is assumed, i.e., we examine for each constraint type if the constraint type depends (1) on the CWA and (2) on the UNA (Section 3).

Using these findings, we are able to determine which constraint types the five most common constraint languages enable to express (see Table 1) The evaluation is explained in detail in the appendix of this paper [5].

Table 1: Expressivity of Constraint Languages

	\mathcal{CT} (81)	$\overline{\mathcal{R}}$ (46)	\mathcal{R} (35)
SPIN	100.0 (81)	100.0 (46)	100.0 (35)
OWL 2 DL	67.9 (55)	45.7 (21)	97.1 (34)
ShEx	29.6 (24)	26.1 (12)	34.3 (12)
ReSh	25.9 (21)	15.2 (7)	40.0 (14)
OWL 2 QL	24.7 (20)	19.6 (9)	31.4 (11)
DSP	17.3 (14)	13.0 (6)	22.9 (8)

We use \mathcal{CT} to refer to the whole set of constraint types, \mathcal{R} to abbreviate the 35 constraint types for which reasoning may be performed before actually validating and $\overline{\mathcal{R}}$ to denote the 46 constraint types for which reasoning does not improve data quality in any obvious sense. For OWL 2, we differentiate between the sub-languages OWL 2 QL and OWL 2 DL as they differ with regard to expressivity and efficiency in performance. Table 1 shows in percentage values (and absolute numbers in brackets) how many \mathcal{CT} , \mathcal{R} , and $\overline{\mathcal{R}}$ constraint types are supported by listed constraint languages. Although OWL 2 is the only language for which reasoning features are already implemented, \mathcal{R} constraint types are also expressible by other languages.

Having information on the constraint type specific expressivity of constraint languages enables validation environments to recommend the right language depending on the users' individual use cases. These use cases determine which requirements have to be fulfilled and therefore which constraint types have to be expressed to meet these use cases. The finding that SPIN is the only language which supports all reasoning constraint types underpins the importance to implement reasoning capabilities by SPIN (or plain SPARQL). The fact that all \mathcal{R} and $\overline{\mathcal{R}}$ constraint types are representable by SPIN emphasizes the significant role SPIN plays for the future development of constraint languages. Another important role may play OWL 2 DL with which 2/3 of all, nearly 1/2 of the $\overline{\mathcal{R}}$, and almost all \mathcal{R} constraint types can be expressed. Even though some \mathcal{R} constraint types correspond to OWL 2 DL axioms, we cannot use them directly to validate RDF data since OWL 2 reasoning and validation assume different semantics which may lead to differences in results.

⁴Online available at: <http://purl.org/net/rdf-validation>

The contributions of this paper are: (1) We identified by today 81 types of constraints that are required by various stakeholders for data applications. (2) We work out the role that reasoning plays in practical data validation, why reasoning is beneficial for RDF validation, and how to overcome the major shortcomings when validating RDF data by performing reasoning prior to validation. (3) For each constraint type, we examine if reasoning may improve data quality, how efficient in terms of runtime validation is performed with and without reasoning, and if validation results depend on the CWA and on the UNA. (4) We determine which constraint types the most common constraint languages enable to express and give directions for the further development of constraint languages. (5) We provide open source validation and reasoning implementations of constraint types to be used to drive the further development of constraint languages (Section 4).

2. REASONING

DL provides the foundational basis for the expressive language *OWL 2* which offers knowledge representation and reasoning services. Validation is not the primary purpose of its design which has lead to claims that *OWL 2* cannot be used for validation. In practice, however, *OWL 2* is well-spread and RDFS/*OWL 2* constructs are widely used to tell people and applications about how valid instances should look like. In general, RDF documents follow the syntactic structure and the semantics of RDFS/*OWL 2* ontologies which could therefore not only be used for reasoning but also for validation.

In this section, we investigate the role that reasoning plays in practical data validation and how to overcome the major shortcomings when validating RDF data by performing reasoning prior to validation. As reasoning is beneficial for validation, we investigate the effect of reasoning to the validation process for each constraint type. Reasoning is beneficial for validation as (1) it may resolve constraint violations, (2) it may cause useful violations, and (3) it solves the redundancy problem. Consider the following DL knowledge base \mathcal{K} - a DL knowledge base is a collection of formal statements which correspond to *facts* or *what is known* explicitly:

```

 $\mathcal{K} = \{$ 
  Book  $\sqsubseteq$  Publication , Book  $\sqsubseteq \forall$  author.Person , Book  $\sqsubseteq \exists$  title.T
  Book(Huckleberry-Finn) , Book(Hamlet) ,
  author(Huckleberry-Finn, Mark-Twain) ,
  title(Huckleberry-Finn, The-Adventures-of-Huckleberry-Finn) ,
  title(Huckleberry-Finn, Die-Abenteuer-des-Huckleberry-Finn) }

```

As we know that books can only have persons as authors (Book $\sqsubseteq \forall$ author.Person), *Huckleberry-Finn* is a book (Book(Huckleberry-Finn)), and *Mark Twain* is its author (author(Huckleberry-Finn, Mark-Twain)), we conclude that *Mark Twain* is a person. As *Mark Twain* is not explicitly defined to be a person, however, a violation is raised. Reasoning may resolve violations (1. benefit). If we apply reasoning before validating, the violation is resolved since the implicit triple Person(Mark-Twain) is inferred and therefore made explicitly available. Reasoning may cause additional violations needed to enhance data quality when these additional violations are resolved (2. benefit). As books are publications (Book \sqsubseteq Publication), constraints on publications are also validated for books which may result in further

valuable violations. As each publication must have a publisher, e.g., a book is a publication, *Huckleberry-Finn* is a book, and *Huckleberry-Finn* does not have a publisher, a violation occurs. This violation would not have been raised without reasoning before actually validating and thus data quality would not be increased in case the violation is tackled. The major shortcoming of classical constraint languages is redundancy. If a particular constraint should hold for multiple classes, it is required to assign the concerned constraint explicitly to each class. The redundancy problem is solved (3. benefit) by associating the constraint with the super-class of these classes and applying *OWL 2* reasoning (see second paragraph in Section 1.1).

Validation environments should enable users to select which constraint types to use for completing data by reasoning and which ones should be considered as constraint types about data accuracy and completeness which could be checked over the data once completed using reasoning. As reasoning is beneficial for validating RDF data, we investigate the effect of reasoning to the validation process for each constraint type, i.e., we examine for each constraint type if reasoning may be performed prior to validation to enhance data quality either (1) by resolving violations or (2) by raising valuable violations. We denote the whole set of constraint types with \mathcal{CT} which we divide into two disjoint sets:

1. \mathcal{R} is the set of constraint types for which reasoning may be performed prior to validation (especially when not all the knowledge is explicit) to enhance data quality either by resolving violations or by raising valuable violations. For \mathcal{R} constraint types, validation is executed by query answering with optional reasoning prior to validation. 35 (43.2%) of the overall 81 constraint types are \mathcal{R} constraint types.
2. $\overline{\mathcal{R}}$ denotes the complement of \mathcal{R} , that is the set of constraint types for which reasoning cannot be done or for which reasoning does not improve data quality in any obvious sense. For $\overline{\mathcal{R}}$ constraint types, validation is performed by query answering without reasoning. 46 (56.8%) of the overall 81 constraint types are $\overline{\mathcal{R}}$ constraint types.

If a journal volume has an *editor* relationship to a person, then the journal volume must also have a *creator* relationship to the same person (*editor* \sqsubseteq *creator*), i.e., *editor* is a sub-property of *creator*. If we use *sub-properties* (R-54/64) without reasoning and the data contains the triple *editor* (A+Journal-Volume, A+Editor), then the triple *creator* (A+Journal-Volume, A+Editor) has to be stated explicitly. If this triple is not present in the data, a violation occurs. If we use *sub-properties* with reasoning, however, the required triple is inferred which resolves the violation. *Sub-properties* is an \mathcal{R} constraint type since reasoning may be performed prior to validation to improve data quality by resolving the violation. *Literal pattern matching* (R-44) restricts literals to match given patterns:

```

1 ISBN a RDFS:Datatype ; owl:equivalentClass [ a RDFS:Datatype ;
2   owl:onDatatype xsd:string ;
3   owl:withRestrictions ([ xsd:pattern "~\d{9}[\d|X|$" ])] .

```

The first *OWL 2* axiom explicitly declares *ISBN* to be a datatype. The second *OWL 2* axiom defines *ISBN* as an abbreviation for a datatype restriction on *xsd:string*. The

datatype *ISBN* can be used just like any other datatype such as in the universal restriction $\text{Book} \sqsubseteq \forall \text{identifier}. \text{ISBN}$ which ensures that books can only have valid *ISBN* identifiers, i.e., strings that match a given regular expression. *Literal pattern matching* is an $\bar{\mathcal{R}}$ constraint type since reasoning cannot be done.

For each constraint type we investigate how efficient in terms of runtime validation is performed with and without reasoning. By mapping to DL we get an idea of the performance of each constraint type in worst case, since the combination of DL constructs needed to express a constraint type determines its computational complexity.

2.1 Constraint Types with Reasoning

\mathcal{R} is the set of constraint types for which reasoning may be performed prior to validation to enhance data quality either by resolving violations or by causing useful violations. For \mathcal{R} constraint types, different types of reasoning may be performed which depends on the language used to formulate the constraint type. 11 of 35 \mathcal{R} constraint types are representable by the less expressive but better performing OWL 2 QL. 23 \mathcal{R} constraint types, in contrast, are not expressible by OWL 2 QL and therefore the more expressive but less performing OWL 2 DL is used. Some of the \mathcal{R} constraint types, however, are also representable by classical constraint languages (e.g., 40% are representable by ReSh). OWL 2 profiles are restricted versions of OWL 2 that offer different trade-offs regarding expressivity vs. efficiency in reasoning. We consider the two extreme OWL 2 profiles, which are OWL 2 QL and OWL 2 DL, as OWL 2 QL is the profile with the highest performance and OWL 2 DL is the profile with the highest expressivity while still being a DL. OWL 2 QL, based on the *DL-Lite* family of DL [2, 6], is an OWL 2 profile which focuses on reasoning in the context of query answering with very large size of instance data. OWL 2 DL was standardized as a DL-like formalism with high expressivity, yet maintains decidability for main reasoning tasks. As a result of its expressive power, OWL 2 DL allows a large variety of sophisticated modeling capabilities for many application domains. The drawback of its expressive power results as a lack of computational efficiency in performance. With regard to the two different types of reasoning we divide \mathcal{R} into two not disjoint sets of constraint types: $\mathcal{R}_{QL} \subseteq \mathcal{R}_{DL}$ (OWL 2 DL is more expressive than OWL 2 QL).

1. \mathcal{R}_{QL} is the set of \mathcal{R} constraint types for which *OWL 2 QL reasoning* may be performed as they are expressible by OWL 2 QL. 11 of 35 \mathcal{R} constraint types are \mathcal{R}_{QL} constraint types.
2. \mathcal{R}_{DL} stands for the set of \mathcal{R} constraint types for which *OWL 2 DL reasoning* may be executed as OWL 2 QL is not expressive enough to represent them [11]. 34 of 35 \mathcal{R} constraint types are \mathcal{R}_{DL} constraint types.

2.1.1 OWL 2 QL Reasoning.

The *property domain* ($R-25$) constraint $\exists \text{author}.\top \sqsubseteq \text{Publication}$ ensures that only publications can have *author* relationships (in OWL 2 QL: `author rdfs:domain Publication`). Without reasoning, the triple `author(Alices-Adventures-In-Wonderland, Lewis-Carroll)` leads to a violation if it is not explicitly stated that *Alices-Adventures-In-Wonderland* is a publication. With reasoning, on the contrary, the class

assignment `rdf:type(Alices-Adventures-In-Wonderland, Publication)` is inferred which prevents the violation to be raised. Thus, reasoning improves data quality by resolving the violation.

The *existential quantification* ($R-86$) $\text{Publication} \sqsubseteq \exists \text{publisher}.\text{Publisher}$ restricts publications to have at least one publisher:

```
1 Publication rdfs:subClassOf
2   [ a owl:Restriction ;
3     owl:onProperty publisher ;
4     owl:someValuesFrom Publisher ] .
```

If reasoning is executed on the triples `publisher (A+Conference-Proceedings, A+Publisher)` and `rdf:type (A+Publisher, Publication)`, it is inferred that *A+Conference-Proceedings* is a publication. Now, all constraints associated with publications are also validated for *A+Conference-Proceedings* - e.g., that publications must have at least one author. Without reasoning, in contrast, the fact that *A+Conference-Proceedings* is a publication is not explicit in the data which is the reason why constraints on publications are not validated for *A+Conference-Proceedings*. Hence, additional violations, which may be useful to enhance data quality in case the violations are taken into account, do not occur.

RDF validation with reasoning corresponds to performing SPARQL queries. As OWL 2 profiles are based on the DL-Lite family, OWL 2 QL is based on *DL-Lite_R*, and query answering in OWL 2 QL is performed in LOGSPACE (or rather in AC^0) [6], the same complexity class applies for validation by queries with reasoning. As TBox reasoning in OWL 2 QL is performed in PTIME [6], complete query rewriting (as well as reasoning and subsequent querying (combined complexity) is carried out in PTIME [2, 6].

2.1.2 OWL 2 DL Reasoning.

Universal quantifications ($R-91$) are used to build anonymous classes containing all individuals that are connected by particular properties only to instances/literals of certain classes/data ranges. Publications, e.g., can only have persons as authors ($\text{Publication} \sqsubseteq \forall \text{author}.\text{Person}$):

```
1 Publication rdfs:subClassOf
2   [ a owl:Restriction ;
3     owl:onProperty author ;
4     owl:allValuesFrom Person ] .
```

When performing reasoning, the triples `author(The-Lord-Of-The-Rings, Tolkien)` and `rdf:type(The-Lord-Of-The-Rings, Publication)` let a reasoner infer that *Tolkien* is a person which satisfies the *universal quantification*. In case reasoning is not executed, a violation is raised since it is not explicitly stated that *Tolkien* is a person. As a consequence, constraints on persons are not checked for *Tolkien* which prevents further validation.

With OWL 2 DL, the more expressive profile than OWL 2 QL, reasoning is executed in N2EXPTIME [11] which is a class of considerably higher complexity than PTIME, the complexity class for OWL 2 QL reasoning. As we consider ontological reasoning, complexity classes are assigned to sets of constraint types according to well-established complexity results in literature on reasoning of DL languages. Therefore, the classification also includes complex logical interferences between TBox axioms.

2.2 Constraint Types without Reasoning

$\overline{\mathcal{R}}$ is the set of constraint types for which reasoning cannot be done or for which reasoning does not improve data quality in any obvious sense. *Context-specific exclusive or of properties* (R-11) is a $\overline{\mathcal{R}}$ constraint type with which it can be defined that an individual of a certain class can either have property A or property B , but not both. Identifiers of publications, e.g., can either be ISBNs (for books) or ISSNs (for periodical publications), but it should not be possible to assign both identifiers to a given publication:

$$\begin{aligned} \mathcal{K} = \{ \\ & \text{Publication} \sqsubseteq (\neg A \sqcap B) \sqcup (A \sqcap \neg B), \\ & A \equiv \exists \text{ isbn.xsd:string}, B \equiv \exists \text{ issn.xsd:string} \end{aligned}$$

This constraint can be represented by OWL 2 DL by building an anonymous class for each exclusive property:

```
1 Publication owl:disjointUnionOf ( A B ) .
2 A rdfs:subClassOf [ a owl:Restriction ;
3   owl:onProperty isbn ;
4   owl:someValuesFrom xsd:string ] .
5 B rdfs:subClassOf [ a owl:Restriction ;
6   owl:onProperty issn ;
7   owl:someValuesFrom xsd:string ] .
```

Exactly the same constraint can be expressed by ShEx more intuitively and concisely:

```
1 Publication { ( isbn xsd:string | issn xsd:string ) }
```

It is a common requirement to narrow down the value space of properties by an exhaustive enumeration of valid values (R-30/37: *allowed values*). Reasoning on this constraint type does not change validation results and therefore does not improve data quality. Books on the topics *Computer Science* and *Librarianship*, e.g., should only have *ComputerScience* and *Librarianship* as subjects. The corresponding DL statement $\text{Book} \equiv \forall \text{ subject.}\{\text{Computer-Science, Librarianship}\}$ is representable by DSP and OWL 2 DL:

```
1 [ a dsp:DescriptionTemplate ;
2   dsp:resourceClass Book ;
3   dsp:statementTemplate [
4     dsp:property subject ;
5     dsp:nonLiteralConstraint [
6       dsp:valueURI ComputerScience, Librarianship ] ] ] .
7
8 subject rdfs:range [ owl:equivalentClass [ a owl:Class ;
9   owl:oneOf ( ComputerScience Librarianship ) ] ] .
```

RDF validation without reasoning corresponds to performing SPARQL queries. It is known that performing SPARQL queries is carried out in PSPACE-Complete [14]. Table 2 gives an overview over the complexity of validation with and without reasoning. The higher the complexity class the worse the performance. The order of the complexity classes is the following [1]:

$$\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE-Complete} \subseteq \text{N2EXPTIME}$$

Table 2: Complexity of Validation According to Reasoning

Validation Type	Complexity Class
$\overline{\mathcal{R}}$	PSPACE-Complete
\mathcal{R}_{QL}	PTIME
\mathcal{R}_{DL}	N2EXPTIME

We do not consider *OWL 2 Full* due to its high worst case complexity (undecidability) and as all (except of one) \mathcal{R} constraint types are already expressible either by OWL 2 QL or OWL 2 DL.

3. CWA AND UNA DEPENDENCY

RDF validation and reasoning assume different semantics. Reasoning in OWL 2 is based on the *open-world assumption* (OWA), i.e., a statement cannot be inferred to be false if it cannot be proved to be true which fits its primary design purpose to represent knowledge on the *WWW*. As each book must have a title ($\text{Book} \sqsubseteq \exists \text{ title.T}$) and *Hamlet* is a book ($\text{Book}(\text{Hamlet})$), *Hamlet* must have at least one title as well. In an OWA setting, this constraint does not cause a violation, even if there is no explicitly defined title, since there must be a title for this book which we may not know (\mathcal{K} is consistent). As RDF validation has its origin in the XML world many RDF validation scenarios require the *closed-world assumption* (CWA), i.e., a statement is inferred to be false if it cannot be proved to be true. Thus, classical constraint languages are based on the CWA where constraints need to be satisfied only by named individuals. In the example, the CWA yields to a violation since there is no explicitly defined title for the book *Hamlet*. OWL 2 is based on the *non-unique name assumption* (nUNA) whereas RDF validation requires that different names represent different objects (*unique name assumption* (UNA)). Although DLs/OWL 2 do not assume UNA, they have the constructs *owl:sameAs* and *owl:differentFrom* to state that two names are the same or different. If validation would assume OWA and nUNA, validation won't be that restrictive and therefore we won't get the intended validation results. This ambiguity in semantics is one of the main reasons why OWL 2 has not been adopted as a standard constraint language for RDF validation in the past.

RDF validation and reasoning assume different semantics which may lead to different validation results when applied to particular constraint types. Hence, we investigate for each constraint type if validation results differ (1) if the CWA or the OWA and (2) if the UNA or the nUNA is assumed, i.e., we examine for each constraint type (1) if the constraint type depends on the CWA and (2) if the constraint type depends on the UNA.

We classify constraint types according to the dependency on the CWA and the UNA which leads to four sets of constraint types: (1) *CWA* denotes the set of constraint types which are dependent on the CWA, i.e., the set of constraint types for which it makes a difference in terms of validation results if the CWA or the OWA is assumed. *Minimum qualified cardinality restrictions* (R-75) is a *CWA* constraint type. Publications, e.g., must have at least one author ($\text{Publication} \sqsubseteq \geq 1 \text{ author.Person}$). In a CWA setting, a publication without an explicitly stated author violates the constraint, whereas, with OWA semantics, a publication without an explicitly stated author does not raise a violation as the constraint entails that there must be an author which we may not know. (2) *CWA* is the complement of *CWA* and thus includes constraint types which are independent on the CWA. Nothing can be a book and a journal article at the same time ($\text{Book} \sqcap \text{JournalArticle} \sqsubseteq \perp$). For the constraint type *disjoint classes* (R-7), it does not make any difference regarding validation results if the CWA or the OWA is taken, as if there is a publication which is a

book and a journal article a violation is raised in both settings, i.e., additional information does not change validation results.

(3) \mathcal{UNA} denotes the set of constraint types which are dependent on the UNA. For *functional properties* (R-57/65), it makes a difference with regard to validation results if the UNA or the nUNA is assumed. As the property *title* is functional (`funct (title)`), a book can have at most one distinct title. UNA causes a clash if the book *Huckleberry-Finn* has more than one title. For nUNA, however, reasoning concludes that the title *The-Adventures-of-Huckleberry-Finn* must be the same as the title *Die-Abenteuer-des-Huckleberry-Finn* which resolves the violation. (4) $\overline{\mathcal{UNA}}$, the complement of \mathcal{UNA} , denotes the set of constraint types which are independent on the UNA. *Literal value comparison* (R-43) is an example of a $\overline{\mathcal{UNA}}$ constraint type which ensures that, depending on property datatypes, two different literal values have a specific ordering with respect to an operator like $<$, $<=$, $>$, and $>=$. It has to be guaranteed, e.g., that birth dates are before ($<$) death dates. If the birth and the death date of *Albert-Einstein* is interchanged (`birthDate(Albert-Einstein, "1955-04-18"), deathDate(Albert-Einstein, "1879-03-14")`), a violation is thrown. The *literal value comparison* constraint type is independent from the UNA as the violation is not resolved in case there are further resources (e.g., *AlbertEinstein*, *Albert_Einstein*) which point to correct birth and death dates and which may be the same as the violating resource when nUNA is assumed.

We evaluated for each constraint type if it is dependent on the CWA and the UNA (for a detailed analysis we refer to the appendix of this paper [5]). The result is that we distinguish between 46 (56.8%) \mathcal{CWA} and 35 (43.2%) $\overline{\mathcal{CWA}}$ and between 54 (66.6%) \mathcal{UNA} and 27 (33.3%) $\overline{\mathcal{UNA}}$ constraint types. Hence, for the majority of the constraint types it makes a difference in terms of validation results if the CWA or the OWA and if the UNA or the nUNA is assumed. For the \mathcal{CWA} and the \mathcal{UNA} constraint types, we have to be careful in case we want to use them for reasoning and for validation as in both usage scenarios we assume different semantics which may lead to different results.

4. IMPLEMENTATION

We use *SPIN*, a SPARQL-based way to formulate and check constraints, as basis to develop a validation environment⁵ to validate RDF data according to constraints expressed by arbitrary constraint languages by mapping them to SPIN⁶ [4]. The SPIN engine checks for each resource if it satisfies all constraints, which are associated with its classes, and generates a result RDF graph containing information about all constraint violations. We provide implementations for all constraint types expressible by OWL 2 QL, OWL 2 DL, and DSP as well as for major constraint types representable by ReSh and ShEx.⁶ By means of a *property ranges* (R-28, R-35) constraint it can be restricted that *author* relations can only point to persons (DL: $\top \sqsubseteq \forall \text{author. Person}$, OWL 2 DL: `author rdfs:range Person`). There is one SPIN construct template for each constraint type, so for the constraint type *property ranges*:

⁵Online available at: <http://purl.org/net/rdfval-demo>, source code available at: <https://github.com/boschthomas/rdf-validator>

⁶SPIN mappings online available at: <https://github.com/boschthomas/rdf-validation/tree/master/SPIN>

```
1 owl2spin:PropertyRanges a spin:ConstructTemplate ;
2   spin:body [ a sp:Construct ; sp:text ""
3     CONSTRUCT {
4       _:cv a spin:ConstraintViolation [...] . }
5     WHERE {
6       ?OP rdfs:range ?C . ?x ?OP ?this . ?this a owl:Thing .
7       FILTER NOT EXISTS { ?this a ?C } . } "" ; ] .
```

A SPIN construct template contains a SPARQL CONSTRUCT query generating constraint violation triples which indicate the subject, the properties, and the constraint causing the violations and the reason why violations have been raised. Violation triples, which are associated with a certain level of severity (informational, warning, error), may also give some guidance how to fix them. A SPIN construct template creates violation triples if all triple patterns within the SPARQL WHERE clause match. If *Doyle*, the author of the book *Sherlock-Holmes* (`author(Sherlock-Holmes, Doyle)`), e.g., is not explicitly declared to be a person, all triple patterns within the SPARQL WHERE clause match and the SPIN construct template generates a violation triple.

Property ranges is an \mathcal{R} constraint type, i.e., a constraint type for which reasoning may be performed prior to validation to enhance data quality. Therefore, validation environments should enable users to decide if reasoning on *property ranges* constraints should be executed before validation. If a user decides to use reasoning, the triple `rdf:type(Doyle, Person)`, whose absence caused the violation, is inferred before data is validated which resolves the violation. Validation environments should enable users (1) to select individual resources for which reasoning should be performed on \mathcal{R} constraints before they are validated, (2) to select \mathcal{R} constraint types for which reasoning should be executed, and (3) to globally determine if for all \mathcal{R} constraint types reasoning should be done. All resources, for which reasoning should be performed prior to validation, are automatically assigned to the class *Reasoning* during a pre-reasoning step. There is one SPIN rule for each \mathcal{R} constraint type, so for *property ranges*:

```
1 owl2spin:Reasoning spin:rule [ a sp:Construct ; sp:text ""
2   CONSTRUCT { ?this a ?C . }
3   WHERE { ?OP rdfs:range ?C . ?x ?OP ?this . ?this a owl:Thing
4     FILTER NOT EXISTS { ?this a ?C } . } "" ; ] .
```

The SPIN rule is executed for each resource of the class *Reasoning*. A SPIN rule contains a SPARQL CONSTRUCT query which generates triples if all triple patterns within the SPARQL WHERE clause match. In case *Doyle* is not defined to be a person, all triple patterns match and the triple `rdf:type(Doyle, Person)` is created. As a consequence, the violation on *Doyle* is not raised. We implemented reasoning capabilities for all \mathcal{R} constraint types for which OWL 2 QL and OWL 2 DL reasoning may be performed.⁶

5. RELATED WORK

Tao [17] suggested an OWL 2 DL extension to support integrity constraints which enables to use OWL 2 as a constraint language for validation under the CWA by conjunctive query answering. Tao also provides a solution to explain and repair integrity constraint violations. Siren and Tao [16] proposed an alternative semantics for OWL 2 using the CWA so that it could be used to validate integrity

constraints. They examined integrity constraint semantics proposed in the deductive databases literature and adopted them for OWL 2 by reducing the validation of integrity constraints to SPARQL query answering by means of reasoners. Although the alternative semantics for OWL 2 is implemented in the *Stardog* database,⁷ it has never been submitted to a standards organization such as the *W3C*.

In DL, reasoning tasks like query answering or detection of inconsistencies require the consideration of knowledge that is not only defined explicitly but also implicitly. To do so there are two different ways called forward- and backward-chaining. The first method implies a materialized knowledge base, where the original knowledge base is extended by all assertions that can be inferred. State-of-the-art DL or OWL reasoners following this approach are *FaCT++* [18], *Pellet* [15], *RacerPro* [8], or *HermiT* [9].

On the second approach, the original knowledge base is kept in its original state. Before queries are evaluated against the knowledge base, queries are rewritten such that the rewritings also consider the implicit knowledge in the result set. Approaches following this way are *PerfectRef* [6] or *TreeWitness* [10], which are implemented in the *ontop*-framework⁸ for ontology-based data access. The first solution is applied on local knowledge bases whereas the second is more appropriate for federative environments like in [12, 13].

6. CONCLUSION AND FUTURE WORK

Based on our work in the DCMI and in cooperation with the W3C working group, we published by today 81 constraint types [5] which form the basis to investigate the role that reasoning and different semantics play for RDF validation.

The conclusions of this paper clearly show that validation results differ depending on whether validation is based on the closed-world assumption (CWA) or the open-world assumption (OWA) and whether the unique name assumption (UNA) or the non-unique name assumption (nUNA) is underlying. Equally, using or not using reasoning has serious impact on which constraints are considered to be violated or fulfilled. Obviously, these findings are not new and should be clear to everyone working with RDF and Semantic Web technologies. According to our experience, however, the topic data validation is yet far too often reduced to the selection of suitable constraint languages which may be related to the obvious but yet inaccurate comparison of RDF with XML as basis technology to represent data. With this paper, we want to make clear that it depends on more than just the constraint language when validating RDF data and when developing appropriate systems. Therefore, in order to realize interoperable solutions for data validation, three components are needed: (1) An adequate constraint language is required that allows to represent the desired constraints. (2) The underlying semantics have to be specified, be it open or closed world, particularly if constraints are used that depend on the choice of the semantics. (3) It must be determined if reasoning should be involved in the validation process or not. Necessary reasoning steps have to be predefined to allow the correct interpretation of the constraints, e.g., when constraints are defined on super-classes to avoid

redundancy.

We investigated the role that reasoning plays in practical data validation and how to overcome the major shortcomings when validating RDF data by performing reasoning prior to validation. Reasoning is beneficial for validation as (1) it may resolve violations, (2) it may cause valuable violations, and (3) it solves the redundancy problem. Users should be enabled to select on which constraint types reasoning should be performed before data is validated and which constraint types to use in order to ensure data accuracy and completeness without reasoning. Therefore, we investigated for each constraint type if reasoning may be performed prior to validation to enhance data quality either by resolving violations or by raising valuable violations. For 43.2% of the constraint types, reasoning may be performed before validating to improve data quality. For 56.8% of the constraint types, however, reasoning cannot be done or does not improve data quality in any obvious sense (Section 2).

For each constraint type, we examined how efficient in terms of runtime validation is performed with and without reasoning in worst case. By mapping constraint types to Description Logics, we were able to determine their computational complexity (Section 2). Validation and reasoning assume different semantics which may lead to different validation results when applied to particular constraint types. OWL 2 reasoning requires the OWA with the nUNA, whereas validation is based on the CWA and the UNA. Therefore, we investigated for each constraint type (1) if it depends on the CWA and (2) if it depends on the UNA. For the majority of the constraint types, it makes a difference in terms of validation results if the CWA or the OWA and if the UNA or the nUNA is assumed (Section 3).

Using these findings, we determined which constraint types the five most common constraint languages enable to express [5]. By revealing which constraint types are not covered by existing languages, we give directions and emphasize the significant role SPIN and OWL 2 DL play for the future development of constraint languages (Section 1.1). SPARQL is generally seen as the method of choice to validate data according to certain constraints [7], although it is not ideal for their formulation. In contrast, constraint languages, which may be placed on top of SPARQL, are comparatively easy to understand and constraints can be formulated more concisely. We use SPIN, a SPARQL-based way to formulate and check constraints, as basis to develop a validation environment⁵ to validate RDF data according to constraints expressed by arbitrary constraint languages by mapping them to SPIN⁶ [4]. We provide open source validation and reasoning implementations of constraint types to be used to drive the further development of constraint languages (Section 4).

As part of future work we extend the *RDF Validator*⁵ to provide a list of languages for which the expressivity is sufficient to represent constraint types depending on users' individual needs. The validation environment may also recommend one of these languages covering the most of the required constraint types with the lowest for the user acceptable complexity class. As reasoning may cause high complexity, the validator may show which constraint types from the users' selections cause the higher complexity class and may provide solutions how to get to the next lower complexity class. It would be charming to have an estimation which group of constraint types demands which complexity class. This is not an easy question, however, since complex-

⁷<http://stardog.com/>

⁸<http://ontop.inf.unibz.it>

ity results are language specific and operational semantics is involved as well. Therefore, it is hard to maintain a general complexity result for a constraint type independent of the language chosen. Yet, providing an estimation for particular cases can still be straightforward.

7. REFERENCES

- [1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.
- [2] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyashev. The DL-Lite Family and Relations. *Journal of Artificial Intelligence Research*, 36(1):1–69, Sept. 2009.
- [3] T. Bosch and K. Eckert. Requirements on RDF Constraint Formulation and Validation. In *Proceedings of the DCMI International Conference on Dublin Core and Metadata Applications*, Austin, Texas, USA, 2014.
- [4] T. Bosch and K. Eckert. Towards Description Set Profiles for RDF using SPARQL as Intermediate Language. In *Proceedings of the DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)*, Austin, Texas, USA, 2014.
- [5] T. Bosch, A. Nolle, E. Acar, and K. Eckert. RDF Validation Requirements - Evaluation and Logical Underpinning. *Computing Research Repository (CoRR)*, abs/1501.03933, 2015.
- [6] D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning*, 39(3):385–429, Oct. 2007.
- [7] C. Fürber and M. Hepp. Using SPARQL and SPIN for Data Quality Management on the Semantic Web. In W. Abramowicz and R. Tolksdorf, editors, *Business Information Systems*, volume 47 of *Lecture Notes in Business Information Processing*, pages 35–46. Springer Berlin Heidelberg, 2010.
- [8] V. Haarslev and R. Müller. RACER System Description. In *Automated Reasoning*, pages 701–705. Springer, 2001.
- [9] I. Horrocks, B. Motik, and Z. Wang. The HermiT OWL Reasoner. In *Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE 2012)*, Manchester, UK, 2012.
- [10] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyashev. The Combined Approach to Ontology-Based Data Access. In *Proceedings of the 22th International Joint Conference on Artificial Intelligence*, volume 3, pages 2656–2661. AAAI Press, 2011.
- [11] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language: Profiles. W3C Recommendation, W3C, 2012.
- [12] A. Nolle, C. Meilicke, H. Stuckenschmidt, and G. Nemirovski. Efficient Federated Debugging of Lightweight Ontologies. In *Web Reasoning and Rule Systems*, pages 206–215. Springer International Publishing, 2014.
- [13] A. Nolle and G. Nemirovski. ELITE: An Entailment-Based Federated Query Engine for Complete and Transparent Semantic Data Integration. In *Proceedings of the 26th International Workshop on Description Logics*, pages 854–867. CEUR Electronic Workshop Proceedings, 2013.
- [14] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16:1–16:45, 2009.
- [15] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.
- [16] E. Sirin and J. Tao. Towards Integrity Constraints in OWL. In *Proceedings of the Workshop on OWL: Experiences and Directions (OWLED 2009)*, 2009.
- [17] J. Tao. *Integrity Constraints for the Semantic Web: An OWL 2 DL Extension*. PhD thesis, Rensselaer Polytechnic Institute, 2012.
- [18] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Automated Reasoning*, pages 292–297. Springer, 2006.