Using Rule-Based Reasoning for RDF Validation

Dörthe Arndt, Ben De Meester, Anastasia Dimou,
Ruben Verborgh, and Erik Mannens

Ghent University - imec - IDLab
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
`doerthe.arndt@ugent.be`

Abstract. The success of the Semantic Web highly depends on its ingredients. If we want to fully realize the vision of a machine-readable Web, it is crucial that Linked Data are actually useful for machines consuming them. On this background it is not surprising that (Linked) Data validation is an ongoing research topic in the community. However, most approaches so far either do not consider reasoning, and thereby miss the chance of detecting implicit constraint violations, or they base themselves on a combination of different formalisms, eg Description Logics combined with SPARQL. In this paper, we propose using Rule-Based Web Logics for RDF validation focusing on the concepts needed to support the most common validation constraints, such as Scoped Negation As Failure (SNAF), and the predicates defined in the Rule Interchange Format (RIF). We prove the feasibility of the approach by providing an implementation in Notation3 Logic. As such, we show that rule logic can cover both validation and reasoning if it is expressive enough.

Keywords: N3, RDF Validation, Rule-Based Reasoning

1 Introduction

The amount of publicly available Linked Open Data (LOD) sets is constantly growing[1], however, the diversity of the data employed in applications is mostly very limited: only a handful of RDF data is used frequently [27]. One of the reasons for this is that the datasets' quality and consistency varies significantly, ranging from expensively curated to relatively low quality data [33], and thus need to be validated carefully before use.

One way to assess data quality is to check them against constraints: users can verify that certain data are fit for their use case, if the data abide to their requirements. First approaches to do that were implementations with hard coded validation rules, such as Whoknows? [13]. Lately, attention has been drawn to formalizing RDF quality assessment, more specifically, formalizing RDF constraints languages, such as Shape Expressions (ShEx) [30] or Resource Shapes (ReSh) [28]. This detaches the specification of the constraints from its implementation.

[1] See, eg statistics at: `http://lod-cloud.net/`.

Constraint languages allow users dealing with RDF data and vocabularies to express, communicate, and test their particular expectations. Such languages can either be (i) existing frameworks designed for different purposes, eg the query language SPARQL [12,21], or the description logic based Web Ontology Language (OWL) [31], or they can be (ii) languages only designed for validation, eg ShEx [30], ReSh [28], Description Set Profiles (DSP) [23], or the forthcoming W3C recommendation candidate Shapes Constraint Language (SHACL) [19]. These different languages can be compared by testing them on commonly supported constraints [9,21], as conducted by Hartmann (né Bosch) et al [8].

Depending on the users' needs, constraint languages have to be able to cope with very diverse kinds of constraints which imply certain *logical requirements*. Such requirements were investigated by Hartmann et al [8] who identified the Closed World Assumption (CWA) and the Unique Name Assumption (UNA) as crucial for validation. Since both are not supported by many Web Logics, Hartmann et al particularly emphasize the difference between reasoning and validation languages and favour SPARQL based approaches for validation which – if needed – can be combined with OWL DL or QL reasoning. In this paper, we take a closer look into these findings from a rule-based perspective: We show that neither UNA nor CWA are necessary for validation if a rule-based framework containing predicates to compare URIs and literals, and supporting Scoped Negation as Failure (SNAF) is used. This enables us to – instead of combining separate, successive systems – do both RDF validation *and* reasoning in only one system which acts directly on a constraint language. We show the feasibility of this approach by providing an implementation. This proof-of-concept is implemented in Notation3 Logic (N3Logic) and tackles the subset of the constraints identified by Hartmann et al [8] which are covered in RDFUnit [21].

The remainder of this paper is structured as follows: In Section 2 we discuss related work. In Section 3 we give an overview of common RDF validation constraints. In Section 4, we discuss how different requirements for RDF validation are met by rule-based logics. Section 5 explains the details of our proof of concept and Section 6 concludes the paper and provides an outlook for future work.

2 Related Work

In this section, first, we will present the state of the art around validation constraint languages. Then, we will give an overview of different languages and approaches used for RDF validation.

Data quality can be described in many dimensions, one of them being the intrinsic dimension, namely, the adherence to a data schema [33]. In the case of RDF data, this implies adhering to certain constraints. These have been carefully investigated by several authors (eg Hartmann et al [12]). The formulation of (a subset of) these constraints can be done using existing languages (eg the Web Ontology Language (OWL) [7], the SPARQL Inferencing Notation (SPIN) [18], or SPARQL [26])), or via dedicated languages (eg Shape Expressions (ShEx) [30], Resource Shapes (ReSh) [28], Description Set Profiles (DSP) [23], or Shapes

Constraint Language (SHACL) [19]. Their execution is either based on reasoning frameworks, or querying frameworks.

On the one hand, Motik et al [22] and Sirin and Tau [29] propose alternative semantics for OWL which support the Closed World Assumption, and are therefore more suited for constraint validation than the original version. To know which semantics apply, constraints have to be marked as such. Using one standard to express both, validation and reasoning, is a strong point of this approach, however, this leads to ambiguity: If the exact same formula can have different meanings, one of the key properties of the Semantics Web – interoperability – is in danger. Another disadvantage of using (modified) OWL as a constraint language is its limited expressiveness. Common constraints such as mathematical operations or specific checks on language tags are not covered by OWL [12].

On the other hand, SPARQL based querying frameworks for validation execution emerged (eg Hartmann [12] or Kontokostas et al [21]). Where Hartmann proposes SPIN as base language to support validation constraints, Kontokostas introduces a similar but distinct language to SPIN, more targeted to validation, so-called Data Quality Test Patterns (DQTP). DQTPs are generalized SPARQL queries containing an extra type of variables. In an extra step, these variables are instantiated based on the RDFS and OWL axioms used by the data schema and can then be employed for querying. As such, the authors assume a closed world semantics for OWL but in contrast to the approaches mentioned above, this special semantics cannot be marked in the ontology itself. They thus change the semantics of the common Web standard OWL. To also find implicit constraint validation an extra reasoning step could be added, but this step would then most probably assume the standard semantics of OWL, further increasing the possibly of experiencing conflicts between the two contradicting versions of the semantics. Hartmann proposes a dedicated ontology to express integrity constraints, and as such, this method does not involve changing existing semantics. For both, involving reasoning is not possible without inclusion of a secondary system.

3 RDF Validation Constraints

Based on the collaboration of the W3C RDF Data Shapes Working Group[2] and the DCMI RDF Application Profiles Task Group[3] with experts from industry, government, and academia, a set of validation requirements has been defined, based on which, 81 types of constraints were published, each of them corresponding to at least one of the validation requirements [9]. This set thus gives a realistic and comprehensive view of what validation systems should support.

Prior to this, the creators of RDFUnit [21] had provided their own set of constraint types they support. Given the usage of RDFUnit in real-world use cases [20], this set gives a good overview of what validation systems should minimally cover.

[2] `https://www.w3.org/2014/data-shapes/wiki/Main_Page`

[3] `http://wiki.dublincore.org/index.php/RDF_Application_Profiles`

Table 1 shows the alignment of the 17 types of constraints as supported by RDFUnit with the relevant constraint types as identified by Hartmann et al [8]. As can be seen, these types are not mapped one-to-one. One constraint from RDFUnit maps to at least one constraint as identified by Hartmann, except for PVT and TRIPLE, which are both not very complex constraints and could thus easily be added to the work of Hartmann et al. In this paper, we mainly focus on these 17 constraints which are all covered by our implementation. To make the topic of constraint validation more concrete, we discuss the examples (TYPEDEP), (INVFUNC) and (MATCH) in more detail later in this paper and refer the interested reader to the above mentioned sources.

4 Features required for Validation

After having listed the kind of constraints relevant for RDF validation in the previous section, we will now focus on the suitability of rule-based logics for that task. Based on the work of Sirin and Tao [29], and Hartmann et al [8] who identified the logical requirements constraint languages need to fulfil, we discuss why rule-based logic is a reasonable choice to validate RDF datasets.

4.1 Reasoning

We start our discussion with reasoning. Hartmann [12, p. 181] points out that performing reasoning in combination with RDF validation brings several benefits: constraint violations may be solved, violations which otherwise would stay undetected can be found, and datasets do not need to contain redundant data to be accepted by a validation engine. To better understand these benefits, consider the following ontology example:

```
:Reseacher rdfs:subClassOf :Person.        (1)
```

And the instance:

```
:Kurt a :Researcher; :name "Kurt01".       (2)
```

If we now have a type dependency constraint (TYPEDEP) saying that every instance of the class `:Researcher` should also be an instance of the class `:Person`, which we test on the data above, a constraint validation error would be raised since `:Kurt` is not declared as a `:Person`. If we perform the same constraint check after reasoning, the triple

```
:Kurt a :Person.                           (3)
```

would be derived and the constraint violation would be solved. Without the reasoning, Triple 3 would need to be inserted into the dataset to solve the constraint, leading to redundant data.

To understand how reasoning can help to detect implicit constraints, consider another restriction: suppose that we have a constraint stating that a person's

Table 1. Constraints Alignment. The first column lists the codes as used in RDFUnit; the second column lists the constraints of Hartmann following the numbering [12, appendix]; and the third column lists the description taken from RDFUnit.

RDFUnit	**Constraint Code**	**Description**
COMP	A11	Comparison between two literal values of a resource
MATCH	A20, A21	A resource's literal value (does not) matches a RegEx
LITRAN	A17, A18	The literal value of a resource (having a certain type) must (not) be within a specific range
TYPEDEP	A4	Type dependency: the type of a resource may imply the attribution of another type
TYPRODEP	A41	A resource of specific type should have a certain property
PVT	B1	If a resource has a certain value V assigned via a property P1 that in some way classifies this resource, one can assume the existence of another property P2
TRIPLE	B2	A resource can be considered erroneous if there are corresponding hints contained in the dataset
ONELANG	A28	A literal value has at most one literal for a language
RDFS-DOMAIN	A13	The attribution of a resource's property (with a certain value) is only valid if the resource is of a certain type
RDFS-RANGE	A14, A15	The attribution of a resource's property is only valid if the value is of a certain type
RDFS-RANGED	A23	The attribution of a resource's property is only valid if the literal value has a certain datatype
INVFUNC	A2	Some values assigned to a resource are considered to be unique for this particular resource and must not occur in connection with other resources
OWL-CARD	A1, A32–37	Cardinality restriction on a property
OWLDISJC	A70	Disjoint class constraint
OWLDISJP	A69	Disjoint property constraint
OWL-ASYMP	A57	Asymmetric property constraint
OWL-IRREFL	A64	Irreflexive property constraint

name should not contain numbers[4]. Without reasoning, no constraint validation would be detected because even though the `:name` of `:Kurt` contains numbers, `:Kurt` would not be detected as an instance of `:Person`.

Hartmann's and many other validation approaches thus suggest to first perform a reasoning step and then do an extra validation step via SPARQL querying. The advantage of using rule-based reasoning instead is that validation can take place during the reasoning process *in one single step*. Relying on a rule which supports **`rdfs:subClassOf`** as for example presented in [2] the aforementioned problem could be detected. In general, OWL-RL [10] can be applied since it is supported by every rule language. If higher complexity is needed, rule languages with support for existential quantification can be used for OWL QL reasoning.

4.2 Scoped Negation as Failure

Another aspect which is important for constraint validation is negation. Hartmann et al claim that the Closed World Assumption is needed to perform validation tasks. Given that most Web logics assume the Open World Assumption, that would form a barrier for the goal of combining reasoning and validation mentioned in the previous section. Luckily, that is not the case. As constraint validation copes with the local knowledge base, Scoped Negation as Failure (SNAF), inter alia discussed in [11,16,25], is enough. Among the logics which support this concept are for example FLORA-2 [14] or N3Logic [6].

In order to understand the idea behind Scoped Negation as Failure, consider the triples that form Formula 2 and suppose that these are the only triples in a knowledge base we want to validate. We now want to test the constraint from above that every individual which is declared as a researcher is also declared as a person (TYPEDEP). This means our system needs to give a warning if it finds an individual which is declared as a researcher, but not as a person:

$$\forall \mathtt{x} : ((\ \mathtt{x\ a\ :Researcher}) \wedge \neg(\ \mathtt{x\ a\ :Person})) \\ \rightarrow (\mathtt{:constraint\ :is\ :violated.}) \quad (4)$$

In the form it is stated before, the constraint cannot be tested with the Open World Assumption. The knowledge base contains the triple

```
:Kurt a :Researcher.
```

but not Triple 3, but the rule is more general: given its open nature, we cannot guarantee that there is no document in the entire Web which declares Triple 3. This changes if we make an addition. Suppose that $\mathcal{K}$ is the the set of triples we can derive (either with or without reasoning) from our knowledge base consisting of Formula 2. Having $\mathcal{K}$ at our disposal, we can test:

$$\forall \mathtt{x} : ((\ \mathtt{x\ a\ :Researcher}) \in \mathcal{K}) \wedge \neg((\ \mathtt{x\ a\ :Person}) \in \mathcal{K})) \\ \rightarrow (\mathtt{:constraint\ :is\ :violated.}) \quad (5)$$

[4] This could be expressed by an extended version of MATCH as for example the constraint "Negative Literal Pattern Matching" in [12].

The second conjunct is not a simple negation, it is a negation with a certain scope, in this case $\mathcal{K}$. If we added new data to our knowledge like for example Triple 3, we would have different knowledge $\mathcal{K}'$ for which other statements hold. The truth value of the formula above would not be touched since this formula explicitly mentions $\mathcal{K}$. The logic stays monotonic. Scoped negation as failure is the kind of negation we actually need in RDF validation: we do not want to make and test statements in the Web in general, we just want to test the information contained in a local file or knowledge base.

4.3 Predicates for Name Comparison

Next to the Open World Assumption, Hartmann et al [8] identify the fact that most Web logics do not base themselves on the Unique Names Assumption (UNA) as a barrier for them being used for constraint validation. This assumption is for example present in F-Logic [17] and basically states that every element in the domain of discourse can only have one single name (URI or Literal in our case). The reason, why this assumption is in general problematic for the Semantic Web lies in its distributed nature: different datasets can – and actually do – use different names for the same individual or concept. For instance, the URI `dbpedia:London` refers to the same place in England as for example `dbpedia-nl:London`. In this case this fact is even stated in the corresponding ontologies using the predicate `owl:sameAs`.

The impact of the Unique Name Assumption for RDF validation becomes clear if we take a closer look at OWL's inverse functional property and the related constraint (INVFUNC). Let us assume that `dbo:capital` is an `owl:InverseFunctionalProperty` and our knowledge base contains:

```
:England dbo:capital :London. :Britain dbo:capital :London.   (6)
```

Since `:England` and `:Britain` are both stated as having `:London` as their capital and `dbo:capital` is an inverse functional property, an OWL reasoner would derive

```
:England owl:sameAs :Britain.   (7)
```

Such a derivation cannot be made if the Unique Name Assumption is valid, since the former explicitly excludes this possibility.

The constraint (INVFUNC) is related to the OWL concept above, but it is slightly different: while OWL's inverse functional property refers to the elements of the domain of discourse denoted by the name, the validation constraint (INVFUNC) refers to the representation itself. Formula 6 thus violates the constraint. Even if our logic does not follow the Unique Name Assumption, this violation can be detected if the logic offers predicates to compare names. In N3Logic, `log:equalTo` and `log:notEqualTo`[5] are such predicates: in contrast to `owl:sameAs` and `owl:differentFrom`, they do not compare the resources

[5] https://www.w3.org/2000/10/swap/doc/CwmBuiltins.

they denote, but their representation. The idea to support these kinds of predicates is very common. So does, for example, the Rule Interchange Format (RIF) cover several functions which can handle URIs and strings, as we will discuss in the next subsection.

4.4 RIF Built-ins

In the previous subsection we indicated that a special predicate of a logic, in this case `log:notEqualTo`, can be used to do URI comparisons and thereby support a concept which would otherwise be difficult to express. Such built-in functions are widely spread in rule-based logics and play an important role in RDF validation which very often deals with string comparisons, calculations or operations on URI level. While it normally depends on the designers of a logic which features are supported, there are also common standards.

One of them is the Rule Interchange Format (RIF) [15] whose aim it is to provide a formalism to exchange rules in the Web. Being the result of a W3C working group consisting of developers and users of different rule-based languages, RIF can also be understood as a reference for the most common features rule based logics might have. This makes the list of predicates [24] supported by the different RIF dialects particularly interesting for our analysis. And it is indeed the case that by only using RIF predicates many of the constraints listed in Section 3 can already be checked: negative pattern matching (MATCH) can be implemented by using the predicate `pred:matches`, the handling of language tags as required for the constraint ONELANG can be done using `func:lang-from-PlainLiteral`, and for the comparison of literal values (COMP) there are several predicates to compare strings, numbers or dates.

To illustrate how powerful RIF is when it comes to string validation, we take a closer look at the predicate `log:notEqualTo` from the previous section. In the example above it is used to compare two URI representations and succeeds if these these two are different. To refer to a URI value, RIF provides the predicate `pred:iri-string` which converts a URI to a string and and vice versa. In N3 notation[6] that could be expressed by:

```
(:England "http://exmpl.com/England") pred:iri-string true.   (8)
```

To compare the newly generated strings, the function `func:compare` can be used. This function takes two string values as input, and returns -1 if the first string is smaller than the second one regarding a string order, 0 if the two strings are the same, and 1 if the second is smaller than the first. The example above gives:

```
("http://exmpl.com/Britain" "http://exmpl.com/England")
                                          func:compare -1.   (9)
```

[6] More about that in Section 5.1

To enable a rule to detect whether the two URI names are equal, one additional function is needed: the reasoner has to detect whether the result of the comparison is not equal to zero. That can be checked using the predicate `pred:numeric-not-equal` which is the RIF version of $\neq$ for numeric values. In the present case the output of the comparison would be `true` since $0 \neq 1$, a rule checking for the name equality of `:England` and `:Britain` using the three predicates would therefore be triggered.

Even though we needed three RIF predicates to express one N3 predicate, the previous example showed how powerful built-ins in general – but also the very common RIF predicates in particular – are. Whether a rule based Web logic is suited for RDF validation highly depends on its built-ins. If it supports all RIF predicates, this can be seen as a strong indication that it is expressive enough.

5 Validation with N3Logic

In the previous section we analysed the requirements on a rule-based Web logic to be able to combine validation and reasoning: it should support scoped negation as failure, it should provide predicates to compare different URIs and strings, and its built-in functions should be powerful enough to, inter alia, access language tags and do string comparison as they are supported by RIF. N3Logic as it is implemented in the EYE reasoner [32] fulfils all these conditions. With that logic, we were able to implement rules for all the constraints listed in Section 3, and thus provide similar functionality as RDFUnit using rule-based Web logics. Below we discuss the details of this implementation starting by providing more information about N3Logic and EYE. The code of our implementation can be accessed at `https://github.com/IDLabResearch/data-validation`.

5.1 N3Logic

N3Logic was introduced in 2008 by Tim Berners-Lee et al [6] and is an extension of RDF: All RDF turtle triples are also valid in N3. As in RDF, blanknodes are understood as existentially quantified variables and the co-occurrence of two triples as in Formula 6 is understood as their conjunction. N3 furthermore supports universally quantified variables. These are indicated by a leading question mark `?`.

```
?x :likes :IceCream.
```
(10)

stands for *"Everyone likes ice cream."*, or in first order logic

$$\forall x : \text{likes}(x, \text{ice-cream})$$

Rules are written using curly brackets `{ }` and the implication symbol `=>`. The `rdfs:subClassOf` relation from Formula 1 can be expressed as:

```
{?x a :Researcher} => {?x a :Person}.
```
(11)

```
@prefix rdfcv: <http://www.dr-thomashartmann.de/phd-thesis/
rdf-validation/vocabularies/rdf-constraints-vocabulary\#>.
@prefix : <http://example.com/constraints#>.
@prefix dbo: <http://dbpedia.org/ontology/> .

:example_constraint a rdfcv:SimpleConstraint;
 :constraintType :InverseFunctionalProperties;
 rdfcv:constrainingElement :inverse-functional-properties;
 rdfcv:leftProperties ( dbo:capital );
 rdfcv:contextClass  dbo:Country.
```

Listing 1. Example inverse functional property constraint: No city can be the capital of two countries.

Applied on Formula 2 the rule results in Formula 3. More details about syntax and semantics of N3 can be found in our previous paper [4].

There are several reasoners supporting N3: FuXi [1] is a forward-chaining production system for Notation3 whose reasoning is based on the RETE algorithm. The forward-chaining cwm [5] reasoner is a general-purpose data processing tool which can be used for querying, checking, transforming and filtering information. EYE [32] is a reasoner enhanced with Euler path detection. It supports backward and forward reasoning and also a user-defined mixture of both. Amongst its numerous features are the option to skolemise blank nodes and the possibility to produce and reuse proofs for further reasoning. The reason why we use EYE in our implementation is its generous support for built-ins[7]: next to N3's native built-ins[8], RIF, but also several other functions and concepts are implemented.

5.2 Expressing Constraints

Before we can detect violations of constraints using N3 logic, these constraints first need to be stated. This could either be done by directly expressing them in rules – and thereby creating a new constraint language next to the ones presented in Section 2 – or on top of existing RDF-based conventions. We opt for the latter and base our present implementation on the work of Hartmann [12, p.167 ff]: in his PhD thesis, Hartmann presents a lightweight vocabulary to describe any constraint, the RDF Constraints Vocabulary (RDF-CV)[9]. The reason why we chose that vocabulary over the upcoming standard SHACL is its expressiveness. We aim to tackle the 81 constraints identified by Hartmann which are not all expressible in SHACL or any other of the constraint languages mentioned in Section 2 [12, p.52, appendix]. As will be shown in the following section, it is not difficult to adopt the rules to different constraint languages as long as they are based on RDF and as such valid N3 expressions.

[7] http://eulersharp.sourceforge.net/2003/03swap/eye-builtins.html
[8] https://www.w3.org/2000/10/swap/doc/CwmBuiltins
[9] https://github.com/boschthomas/RDF-Constraints-Vocabulary

RDF-CV supports the concept of so called *simple constraints* which are all the constraints expressible by the means of the vocabulary, in particular the ones mentioned in Section 3. Each simple constraint has a constraining element. Where applicable, the names of these elements are inspired by their related DL names, but the constraining element can also be for example the name of a SPARQL function. In some cases, the same constraint type can be marked by different constraining elements as for example the constraint COMP whose constraining element is the relation used to compare values (eg the usual numerical orders: $<, >, \leq$, and $\geq$) or there can be different constraint types sharing the same constraining element. To be sure that cases like this do not cause any ambiguity we additionally assign a *constraint type* to every constraint. The names of these types follow the names used by Hartmann [12, appendix]. The TYPEDEP constraint from Section 4.1 is for example of constraint type `:Subsumption`.

In addition to constraining element and constraint type, there are several predicates to assign the constraints to individuals and classes: *context class*, *classes*, *leftProperties*, *rightProperties*, and *constraining values*. The *context class* of a constraint fixes the set of individuals for which a constraint must hold. For the subsumption constraint mentioned above, that would be the class `:Researcher`, the constraint talks about *every* individual labelled as *researcher*. There could be other classes involved. In our subsumption example that is the superclass the individuals should belong to, `:Person`. Every researcher should also be labelled as *person*. Since these kinds of properties can be multiple, they are given in a list. How and if the predicate *classes* is used depends on the constraint. The predicates *leftProperties* and *rightProperties* are used to do similar statements about properties. The constraint INVFUNC as displayed in Listing 1 makes for example use of it to relate the constraint specified to the predicate it is valid for. The objects of the predicates *leftProperties* and *rightProperties* are lists. The predicate *constraining value* is used for the predicates where a literal value is needed to further specify a constraint. An example for such a constraint is MATCH as described in Section 4.1. To express, that a name should not contain numbers, the predicate *constraining value* connects the constraint to the string pattern, `"[1-9]"` in the present case.

5.3 Constraint Rules

Having seen in the last section one possible way to describe constraints on RDF datasets, this section explains how these descriptions can be used. We employ rules which take the expressed constraints and the RDF dataset to be tested into account and generate triples indicating constraint validations, if present. We illustrate that by an example: In Listing 2 we provide a rule handling the constraint INVFUNC. Lines 7–11 contain the details of the constraint. The rule applies for simple constraints of the type *inverse functional properties* for which a context class `?Class` and a list `?list` of left properties is specified. This part of the rule's antecedence unifies with the constraint given in Listing 1. Lines 13–18 describe which situation in the tested data causes a constraint violation: for an `?object` which is an instance of `?Class`, there are two subjects, `?x1` and `?x2`,

```
 1 @prefix rdfcv: <http://www.dr-thomashartmann.de/phd-thesis/
   rdf-validation/vocabularies/rdf-constraints-vocabulary\#> .
 2 @prefix : <http://example.com/constraints#> .
 3 @prefix list: <http://www.w3.org/2000/10/swap/list#>.
 4 @prefix log: <http://www.w3.org/2000/10/swap/log#> .
 5
 6 {
 7 ?constraint a rdfcv:SimpleConstraint;
 8  :constraintType :InverseFunctionalProperties;
 9  rdfcv:constrainingElement :inverse-functional-properties;
10  rdfcv:leftProperties ?list;
11  rdfcv:contextClass  ?Class.
12
13 ?object a ?Class.
14 ?property list:in ?list.
15 ?x1 ?property ?object.
16 ?x2 ?property ?object.
17 ?x1 log:notEqualTo ?x2
18 }
19 =>
20 {
21 [] a :constraintViolation;
22     :violatedConstraint ?constraint.
23 }.
```

Listing 2. Rule for inverse functional property (INVFUNC). The predicate `log:notEqualTo` compares the resources based on their URI and thereby supports the Unique Name Assumption.

defined which are both connected to `?object` via `?property`. This `?property` is an element of `?list`, and the names, ie the URI- or string-representations, of `?x1` and `?x` differ. The latter is expressed using the predicate `log:notEqualTo`[10] (Line 18). Together with Listing 1 that violation is thus detected if two different resource names for resources of the class `dbo:Country` are connected via the predicate `dbo:capital` to the same object. Assuming that `:Britain` and `:England` are both instances of the class `dbo:Country`, the triples in Formula 6 lead to the violation:

$$\texttt{\_:x a :violaton; :violatedConstraint :example\_constraint.} \tag{12}$$

The example shown relies on descriptions following the vocabulary Hartmann suggests, but our approach can easily be adapted for other RDF based constraint vocabularies. All we need is a consistent way to express constraints in RDF. Note that our rules act directly on constraint descriptions and RDF datasets: while the SPARQL based approaches [12,21] mentioned in Section 2 rely on an extra mapping step to instantiate the search patterns. If reasoning needs to be

[10] As explained in Section 4.4 there are alternative ways to express the predicate `log:notEqualTo` in N3, the antecedence of the entire rule could also be expressed only using RIF predicates.

included into the data validation, a rule based system can do reasoning, mapping and constraint validation in one single step where other systems need to perform three.

6 Conclusion and Future Work

In this paper we investigated the requirements a rule based Web logic needs to fulfil to be suitable for RDF validation: it should support Scoped Negation as Failure, it should provide predicates for name comparison, and its built-ins should be powerful enough to, for example, do string comparisons or access language tags. Together with the capability to meet its primary purpose, Web reasoning, such a Web logic is a strong alternative to the common approach of either combining reasoning and validation in two different steps, for example by first performing OWL reasoning and then executing SPARQL queries on top of the result as done by Hartmann [12], or only executing SPARQL queries and thereby ignoring possible implicit constraint violations as done in RDFUnit [21]. Rule based Web logics fulfilling the requirements still provide the same expressivity as SPARQL with the additional advantage of supporting reasoning. Validation and reasoning can thus be done by *one single system* in *one single step*. The practical feasibility of this approach has been shown by providing a proof-of-concept in N3Logic which supports all RDFUnit constraint types. As such, we allow users to assess their data quality more easily using a single rule based validation system, and potentially uncovering more errors. Thus, improving data quality on the Semantic Web overall.

In future work, we are planning to extend our implementation: we aim to cover all of the 81 constraints identified by Hartmann et al [8] which are not specific to SPARQL. We furthermore envisage to extend the supported RDF constraint vocabularies and to align our efforts with SHACL. Another direction of future research will be a better combination of performant reasoning and validation, following the ideas provided in previous work [3]. Further evaluation on performance is also to be conducted.

Acknowledgements The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union, in the context of the project “DiSSeCt”, which is a collaboration by SMIT, DistriNet, and IDLab.

References

1. FuXi 1.4: A Python-based, bi-directional logical reasoning system for the semantic web, `http://code.google.com/p/fuxi/`
2. Arndt, D., De Meester, B., Bonte, P., Schaballie, J., Bhatti, J., Dereuddre, W., Verborgh, R., Ongenae, F., De Turck, F., Van de Walle, R., Mannens, E.: Ontology

reasoning using rules in an eHealth context. In: Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D. (eds.) Proceedings of the 9th International RuleML Symposium. Lecture Notes in Computer Science, vol. 9202, pp. 465–472. Springer (Jul 2015), `http://link.springer.com/chapter/10.1007/978-3-319-21542-6_31`

3. Arndt, D., De Meester, B., Bonte, P., Schaballie, J., Bhatti, J., Dereuddre, W., Verborgh, R., Ongenae, F., De Turck, F., Van de Walle, R., Mannens, E.: Improving OWL RL reasoning in N3 by using specialized rules. In: Tamma, V., Dragoni, M., Gonçalves, R., Ławrynowicz, A. (eds.) Ontology Engineering: 12th International Experiences and Directions Workshop on OWL. Lecture Notes in Computer Science, vol. 9557, pp. 93–104. Springer (Apr 2016), `http://dx.doi.org/10.1007/978-3-319-33245-1_10`
4. Arndt, D., Verborgh, R., De Roo, J., Sun, H., Mannens, E., Van de Walle, R.: Semantics of Notation3 logic: A solution for implicit quantification. In: Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D. (eds.) Rule Technologies: Foundations, Tools, and Applications. Lecture Notes in Computer Science, vol. 9202, pp. 127–143. Springer (Jul 2015), `http://link.springer.com/chapter/10.1007/978-3-319-21542-6_9`
5. Berners-Lee, T.: *cwm* (2000–2009), `http://www.w3.org/2000/10/swap/doc/cwm.html`
6. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008)
7. Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M.: OWL 2 Web Ontology Language. W3C Recommendation (Dec 2012), `http://www.w3.org/TR/owl2-syntax/`
8. Bosch, T., Acar, E., Nolle, A., Eckert, K.: The role of reasoning for rdf validation. In: Proceedings of the 11th International Conference on Semantic Systems. pp. 33–40. SEMANTICS '15, ACM, New York, NY, USA (2015), `http://doi.acm.org/10.1145/2814864.2814867`
9. Bosch, T., Nolle, A., Acar, E., Eckert, K.: Rdf validation requirements-evaluation and logical underpinning. arXiv preprint arXiv:1501.03933 (2015)
10. Calvanese, D., Carroll, J., Di Giacomo, G., Hendler, J., Herman, I., Parsia, B., Patel-Schneider, P.F., Ruttenberg, A., Sattler, U., Schneider, M.: OWL 2 Web Ontology Language Profiles (second edition). W3C Recommendation (Dec 2012), `www.w3.org/TR/owl2-profiles/`
11. Damásio, C.V., Analyti, A., Antoniou, G., Wagner, G.: Supporting Open and Closed World Reasoning on the Web, pp. 149–163. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), `http://dx.doi.org/10.1007/11853107_11`
12. Hartmann, T.: Validation Framework for RDF-based Constraint Languages. Ph.D. thesis, Dissertation, Karlsruhe, Karlsruher Institut für Technologie (KIT), 2016 (2016)
13. Ketterl, M., Knipping, L., Ludwig, N., Mertens, R., Waitelonis, J., Ludwig, N., Knuth, M., Sack, H.: Whoknows? evaluating linked data heuristics with a quiz that cleans up dbpedia. Interactive Technology and Smart Education 8(4), 236–248 (2011)
14. Kifer, M.: Nonmonotonic Reasoning in FLORA-2, pp. 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), `http://dx.doi.org/10.1007/11546207_1`
15. Kifer, M.: Rule interchange format: The framework. In: International Conference on Web Reasoning and Rule Systems. pp. 1–11. Springer (2008)

16. Kifer, M., de Bruijn, J., Boley, H., Fensel, D.: A Realistic Architecture for the Semantic Web, pp. 17–29. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11580072_3
17. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM 42(4), 741–843 (Jul 1995), http://doi.acm.org/10.1145/210332.210335
18. Knublauch, H., Hendler, J.A., Idehen, K.: SPIN – overview and motivation. Tech. rep., W3C (Feb 2011), https://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/, accessed April 18th, 2016
19. Knublauch, H., Kontokostas, D.: Shapes constraint language (shacl). Tech. rep., W3C (2017), https://www.w3.org/TR/shacl/, accessed March 3rd, 2017
20. Kontokostas, D., Mader, C., Dirschl, C., Eck, K., Leuthold, M., Lehmann, J., Hellmann, S.: Semantically enhanced quality assurance in the jurion business use case. In: 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 2016. pp. 661–676 (2016), http://svn.aksw.org/papers/2016/ESWC_Jurion/public.pdf
21. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.: Test-driven evaluation of linked data quality. In: Proceedings of the 23rd international conference on World Wide Web. pp. 747–758. ACM (2014)
22. Motik, B., Horrocks, I., Sattler, U.: Adding integrity constraints to owl. In: OWLED. vol. 258 (2007)
23. Nilsson, M.: Description set profiles: A constraint language for dublin core application profiles. DCMI Working Draft (2008)
24. Polleres, A., Boley, H., Kifer, M.: Rif datatypes and built-ins 1.0 (second edition). W3C Recommendation (Feb 2013), https://www.w3.org/TR/rif-dtb/
25. Polleres, A., Feier, C., Harth, A.: Rules with Contextually Scoped Negation, pp. 332–347. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), http://dx.doi.org/10.1007/11762256_26
26. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (Jan 2008), http://www.w3.org/TR/rdf-sparql-query/
27. Rietveld, L., Beek, W., Schlobach, S.: LOD lab: Experiments at LOD scale. In: International Semantic Web Conference. pp. 339–355. Springer (2015)
28. Ryman, A.: Resource shape 2.0. w3c member submission. World Wide Web Consortium, Feb (2014)
29. Sirin, E., Tao, J.: Towards integrity constraints in owl. In: Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529. pp. 79–88. OWLED'09, CEUR-WS.org, Aachen, Germany, Germany (2009), http://dl.acm.org/citation.cfm?id=2890046.2890055
30. Solbrig, H., Prud'hommeaux, E.: Shape expressions 1.0 definition. w3c member submission. World Wide Web Consortium, June (2014)
31. Tao, J.: Integrity constraints for the semantic web: an owl 2 dl extension. Ph.D. thesis, Rensselaer Polytechnic Institute (2012)
32. Verborgh, R., De Roo, J.: Drawing conclusions from linked data on the web. IEEE Software 32(5) (May 2015), http://online.qmags.com/ISW0515?cid=3244717&eid=19361&pg=25
33. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment for linked data: A survey. Semantic Web 7(1), 63–93 (Mar 2015), http://www.semantic-web-journal.net/system/files/swj556.pdf