RDF Graph Validation Using Rule-Based Reasoning

Ben De Meester [a,*], Pieter Heyvaert [a], Dörthe Arndt [a], Anastasia Dimou [a], and Ruben Verborgh [a]
[a] *Ghent University – imec – IDLab, Department of Electronics and Information Systems, Technologiepark-Zwijnaarde 122, 9052 Ghent, Belgium*
E-mails: ben.demeester@ugent.be, pheyvaer.heyvaert@ugent.be, doerthe.arndt@ugent.be, anastasia.dimou@ugent.be, ruben.verborgh@ugent.be

Abstract. The correct functioning of Semantic Web applications requires that given RDF graphs adhere to an expected shape. This shape depends on the RDF graph and the application's supported entailments of that graph. During validation, RDF graphs are assessed against sets of constraints, and found violations help refining the RDF graphs. However, existing validation approaches cannot always explain the root causes of violations (inhibiting refinement), and cannot fully match the entailments supported during validation with those supported by the application. These approaches cannot accurately validate RDF graphs, or combine multiple systems, deteriorating the validator's performance. In this paper, we present an alternative validation approach using rule-based reasoning, capable of fully customizing the used inferencing steps. We compare to existing approaches, and present a formal ground and practical implementation "Validatrr", based on N3Logic and the EYE reasoner. Our approach – supporting an equivalent number of constraint types compared to the state of the art – better explains the root cause of the violations due to the reasoner's generated logical proof, and returns an accurate number of violations due to the customizable inferencing rule set. Performance evaluation shows that Validatrr is performant for smaller datasets, and scales linearly w.r.t. the RDF graph size. The detailed root cause explanations can guide future validation report description specifications, and the fine-grained level of configuration can be employed to support different constraint languages. This foundation allows further research into, a.o., handling recursion, validating RDF graphs based on their generation description, and providing automatic refinement suggestions.

Keywords: Constraints, Rule-based Reasoning, Validation

1. Introduction

Semantic Web data is represented using the Resource Description Framework (RDF), forming an *RDF graph* [20]. The *quality* of an RDF graph – its "fitness for use" [74] – heavily influences the results of a Semantic Web *application* [51]. An RDF graph's fitness for use depends on its *shape* i.e., the RDF graph itself and the application's *supported entailments* of that RDF graph. For example, some applications support inferring `rdfs:subClassOf` entailments [16], whereas other applications require the RDF graph to explicitly contain all classifying triples (i.e., `rdfs:subClassOf` entailment is not supported).

[*]Corresponding author. E-mail: ben.demeester@ugent.be.

RDF graphs are *validated* by assessing their adherence to a set of *constraints* [50], and different applications (i.e., different *use cases*) specify different sets of constraints. Via validation, we discover (portions of) RDF graphs that do not conform to these constraints, i.e., the *violations* that occur. These violations guide the user to the resources and relationships related to the constraints. *Refining* these resources and relationships results in an RDF graph of higher quality [26], thus, RDF graph validation is an important element for the correct functioning of Semantic Web applications.

1.1. Validation problems

Let us consider the following example: an RDF graph containing people and their birthdates is validated. The use case dictates the set of constraints and the supported entailments. Specifically, we validate formula (1)[1], with a relevant ontology represented in formula (2).

```
:Bob :firstname "Bob" ;                          (1)
     :birthdate "1970-01-01"^^xsd:date .
:birthdate rdfs:domain :Person .                 (2)
:Bob a :Person .                                 (3)
```

Problem 1 (P1): finding the root causes of violations. For example, the use case dictates following compound constraint $c_{compound}$: given a resource r, this resource has $(r_{firstname} \wedge r_{lastname}) \vee (r_{nickname})$. When the RDF graph contains formula (1) and formula (3), `:Bob` should be marked as a resource violating $c_{compound}$. However, when refining the RDF graph, the *root cause* of the violation is needed: does the resource lack firstname, lastname, or nickname?

For constraint types such as compound constraints, existing validation approaches typically return the resource that violates the constraint. However, more detailed descriptions are typically not provided, and manual inspection is needed to discover the root cause: *why* a resource violates a constraint. Without the root cause, it is hard to (automatically) refine the RDF graph and improve its quality.

Problem 2 (P2): the number of found violations depends on the supported entailments. A mismatch between which entailments are supported during validation and which entailments are supported by the use case influences, e.g., whether formula (3) is inferred or not. Thus, either too many or too few violations can be returned [13]. This difference in number of found violations gives a biased idea of the real quality of the validated RDF graph.

Too many violations: formula (2) specifies the domain of `:birthdate`. Let us validate that "every resource in the RDF graph that has a birthdate, is a person" given formula (1). When the entailments of formula (2) are not supported, this would result in a violation: formula (3) is missing in the RDF graph. However, when the entailments of formula (2) are supported, we can infer formula (3), and no violation is returned.

Too few violations: Let us validate that "every person in the RDF graph adheres to constraint $c_{compound}$" given formula (1). Formula (3) is not explicitly stated and the entailments of formula (2) are not supported. No violations are found: `:Bob` is not explicitly classified as a `:Person`, thus `:Bob` is not targeted by $c_{compound}$. However, supporting those entailments can create new statements to be validated, and lead to new violations. For example, by inferring formula (3) using formula (2), `:Bob` is targeted by – and violates – $c_{compound}$. Such violations are not found in the original RDF graph, but discovered due to the supported entailments.

Customizing the *set of inferencing steps* during validation (e.g., whether `rdfs:domain` entailments are supported or not) allows to match the entailments supported by the use case with those of the validation approach. However, support for customizable inferencing steps is limited. When a fixed set (or no set) of inferencing steps is supported, a separate reasoning process is needed to infer unsupported entailments, and edge cases handling this fixed set cannot be validated accurately. For example, let us look at the W3C recommended Shapes Constraint Language (SHACL): a language for validating RDF graphs against a set of constraints [47]. SHACL specifies a fixed set of inferencing steps during validation, namely, `rdfs:subClassOf` entailment. Thus, one cannot validate, e.g., whether an RDF graph explicitly contains all triples that link resources to all their classes given a set of `rdfs:subClassOf` axioms, as `rdfs:subClassOf` triples are always inferred by a conform SHACL validator. RDF graphs that do not contain all classifying triples will be valid according to SHACL validators, however, they are handled poorly by applications that do not support `rdfs:subClassOf` entailment.

Problem 3 (P3): Combining validation with a reasoning preprocessing step decreases performance Entailments can be inferred by performing reasoning as a preprocessing step prior to validation [13], thus combining multiple systems. The resulting RDF graph then explicitly contains all supported entailments, given that the reasoner can be configured to only infer the entailments that are supported by the use case. The number of found violations is then accurate with respect to the use case (solving P2). However, this requires a sequence of independent systems. Thus, the preprocessing step possibly produces entailments not relevant for validation [13]. This independent generation of unnecessary entailments can decrease the performance compared to a single validation system. More, due to this sequence of independent systems, finding the root causes involves investigating the results of both systems: the

[1]For the remainder of the paper, empty prefixes denote the fictional schema http://example.com/, other prefixes are conform with the results of https://prefix.cc.

validator who detects violations, and the reasoner who infers entailments.

1.2. Hypotheses

To solve aforementioned observed validation problems, we pose following hypotheses:

Hypothesis 1: root causes can be explained more accurately compared to existing validation approaches when using a declarative logic.

Hypothesis 2: a more accurate number of violations are found compared to existing validation approaches when supporting a custom set of inferencing steps.

Hypothesis 3: a validation approach supporting more accurate root cause explanations and a custom set of inferencing steps can support an equivalent number of constraint types compared to existing approaches.

Hypothesis 4: a validation approach supporting a custom set of inferencing steps is faster than an approach that includes the same inferencing as a preprocessing step.

1.3. Contributions

In this paper, we propose an approach for RDF graph validation that uses a rule-based reasoner as its underlying technology. Rule-based reasoners can generate a proof stating which rules were triggered for which returned violation. Thus, the root causes of violations can be accurately explained (*solving P1*).

A validation approach using rule-based reasoning natively support the inclusion of a custom set of inferencing steps by adding custom rules. The supported entailments during validation can thus be matched to the entailments supported by the use case, and the validation returns an accurate number of found violations (*solving P2*).

Moreover, rule-based reasoners only need a single language to declare both the constraints and the set of inferencing rules, and only a single system to execute the validation. Compared to a combination of a reasoner and a validation system, this approach does not lead to the generation of entailments unnecessary to the validation step, making it potentially faster than including an inferencing preprocessing step (*solving P3*).

Our contributions are as follows:

i An analysis of existing validation approaches and comparison to a rule-based reasoning approach.
ii A formal ground for using rule-based reasoning for validation.
iii An implementation using N3Logic [8] to define the inferencing and validation rules, executed using the EYE reasoner [73], supporting general constraint types as described by Hartmann et al. [35].
iv An evaluation of our approach, positioning it within the state of the art by functionally validating the hypotheses and comparing the validation speed.

We validated that (a) the formal logical proof explains the root cause of a violation more detailed than the state of the art; (b) an accurate number of violations is returned by using a custom set of inferencing rules up to at least OWL-RL complexity and expressiveness; (c) the number of supported constraint types is equivalent to existing validation approaches; and (d) our implementation is faster than a combined system, and faster than an existing validation approach when RDF graphs are smaller than one hundred thousand triples.

The remainder of the paper is organized as follows: We give an overview of the state of the art (Section 2), after which we position and compare rule-based reasoning as validation approach (Section 3). We provide a formal ground (Section 4) and practical implementation (Section 5), evaluate our proposed approach (Section 6), and summarize our conclusions (Section 7).

2. State of the art

In this work, we propose an alternative validation approach using rule-based reasoning. We first provide a background on validation and reasoning in Section 2.1. Then, we give an overview of existing validation approaches in Section 2.2, and of related vocabularies and ontologies in Section 2.3. We conclude with an overview of general constraint types in Section 2.4, which allows us to functionally compare validation approaches. Our categorization is derived from the general quality surveys of Zaveri et al. [74], Ellefi et al. [28], and Tomaszuk [72], and from the "Validating RDF Data" book [51]. The related works are extended with recent works published in, a.o., the major Semantic Web conferences (ESWC and ISWC), and the major Semantic Web journals (Journal of Web Semantics and Semantic Web Journal).

2.1. Background

Validation Data quality can be assessed by employing a set of *data quality assessment metrics* [9]. Quality assessment for the Semantic Web – and more specifically, for Linked Data – spans multiple dimensions, further categorized in *accessibility*, *intrinsic*, *trust*, *dataset dynamicity*, *contextual*, and *representational* dimensions [74]. Validating an RDF graph directly relates to intrinsic quality dimensions, as defined by Zaveri et al. [74]: (i) independent of the user's context, and (ii) checking if information correctly and compactly represents the real world data and is logically consistent in itself, i.e., the graph's adherence to a certain schema or shape. In this paper, we specifically focus on RDF graph validation, i.e., the intrinsic dimensions.

Validation of an RDF graph can be automated by using a set of *test cases*, each assessing a specific *constraint* [50]. *Violations* of those constraints are then indicated when a validation returns negative results. Validation is typically achieved following Closed World Assumption (CWA): what is not known to be true must be false. For example, a validation assesses for a specific RDF graph if all objects linked via the predicate `schema:birthdate` are a valid `xsd:date`, or if all subjects and objects linked via the predicate `foaf:knows` are explicitly listed to be of type `:Human`. Negative results are returned, indicating violations.

Reasoning Ontologies are prevalent in the Semantic Web community to represent the knowledge of a domain. *Ontology languages* are used to annotate asserted facts (*axioms*). Examples include RDF Schema (RDFS) [16] and the Web Ontology Language (OWL) [39]. *Reasoning* on top of these axioms is achieved, as the calculus of the used *logic* specifies a set of *inferencing steps*, inferring logical consequences (*entailments*) from these axioms [24]. Semantic Web logics – given the open nature of the Web – typically follow the Open World Assumption (OWA): what is not known to be true is simply unknown.

Semantic Web reasoners are typically *description logic-based reasoners* supporting OWL-DL or subprofiles such as OWL-QL [55], or *rule-based reasoners* [59]. Description logic-based reasoners are typically optimized for specific description logics, such as KAON2[2] for $\mathcal{SHIQ}$ and FaCT++[3] for $\mathcal{SROIQ}$. Rule-based reasoners typically follow two types of inferencing algorithms: *forward chaining* and *backward chaining* [59]. Whereas forward chaining tries to infer as much new information as possible, backward chaining is goal-driven: the reasoner starts with a list of goals and tries to verify whether there are statements and rules available that support any of these goals [59]. The employed rules define the logic followed by rule-based reasoners such as EYE [73] or cwm [6]. Whereas description logic-based reasoners have (optimized) inferencing steps for, e.g., `rdfs:subClassOf` and other RDFS or OWL constructs embedded, rule-based reasoners commonly rely on the general "implies" construct. Each rule specifies "A implies B", where both the *antecedent* "A" and the *consequence* "B" can consist of statements [59]. Certain constructs such as `rdfs:subClassOf` can be translated into one or more rules[4].

There is a clear distinction between ontologies and the constraint set for RDF graph validation: ontologies focus on the representation of a domain, whereas RDF graph validation checks whether the resources of that graph conform to a desired schema [51]. It is not required that the representation of a domain aligns with the schema for validation. However, they can complement each other. The usage of ontologies prescribes a set of inferencing steps, for example, the FOAF ontology declares the `rdfs:range` of the `foaf:knows` predicate as `foaf:Person` [17]. Whether these inferencing steps are taken into account during validation or not, influences the number of found violations [13].

2.2. Validation Approaches

In this section, we discuss RDF graph validation approaches. Tools and surveys that cover quality dimensions other than the intrinsic dimensions such as accessibility or representational dimensions are out of scope. We discuss the approaches roughly in chronological order: *hard-coded*, using *integrity constraints*, *query-based*, and using a *high-level language*. Except from *hard-coded* systems, these validation approaches propose or use some kind of declarative means to describe RDF graph constraints.

2.2.1. Hard-coded

Hard-coded systems are a black box where the business logic lies within the code base: the implementation embeds both description and validation of constraints. Hogan et al. analyzed common quality problems both for publishing and intrinsic quality dimensions [40], providing an initial set of best practices [41]. Efforts focus on a limited set of configurable settings (i.e., turning constraint rules on or off) [53].

[2]http://kaon2.semanticweb.org/
[3]http://owl.cs.manchester.ac.uk/tools/fact/
[4]http://eulersharp.sourceforge.net/#theories

2.2.2. *Integrity Constraints*

For these validation approaches, the axioms of vocabularies and ontologies used by the validated RDF graph are *interpreted as integrity constraints* [54, 60, 71]. For example, disjointness forces a description logic-based reasoner to throw an error, which is interpreted as a violation. To combine CWA typically assumed for validation with OWA assumed in ontology languages, alternative semantics for these ontology languages are proposed. The underlying technology used is a description logic-based reasoner or a SPARQL endpoint.

Description logic-based reasoner Motik et al. [54] propose semantic redefinitions, where a certain subset of axioms are designated as constraints. To know which alternative semantics for OWL apply, constraints have to be marked as such. They propose to integrate their implementation with KAON2. Furthermore, custom integrity constraints for Wordnet have been verified using Protégé [56] with FaCT++ [18].

SPARQL endpoint Tao et al. [71] propose using OWL expressions with Closed World assumption and a weak variant of Unique Name assumption to express integrity constraints. OWL semantics are redefined, without being explicitly stated as such during validation. They use SPARQL [1] for axioms described in RDF, RDFS, and OWL [71], e.g., using SPARQL property paths to simulate `rdfs:subClassOf` entailment. Tao et al. work in a general OWL setting, where their approach is sound but not complete. In an RDF setting the approach is both sound and complete, as there is only a single model that needs to be considered [60]. This implementation is incorporated into Stardog ICV [63]. Patel-Schneider separates validation into integrity constraints and Closed World recognition [60], showing that RDF and RDFS entailment can be implemented for both by translation to SPARQL queries.

2.2.3. *Query-based*

In query-based approaches, constraints are described and interpreted similar to SPARQL queries [61]: only RDF graphs whose structure is compatible with the defined structure are returned. These approaches use an embedded or external SPARQL endpoint as underlying technology.

CLAMS [29] is a system to discover and resolve inconsistencies in Linked Data. They define a violation as a minimal set of triples that cannot coexist. The system identifies all violations by executing a SPARQL query set. Knublauch et al. propose the SPARQL Inference Notation (SPIN) [48]: a SPARQL-based rule and constraint language. The SPARQL query is described using RDF statements instead of using the original SPARQL syntax. Kontokostas et al. [50] propose Data Quality Test Patterns (DQTP): tuples of typed pattern variables and a SPARQL query template to declare test case patterns. The validation framework that validates these DQTPs is called RDFUnit. The DQTPs are transformed into SPARQL queries, where every SPARQL query is a test case. RDFUnit additionally allows automatically generated test cases, depending on the used schema.

RDFUnit is also used to validate Linked Data generation rules in the RDF Mapping Language (RML) [25], by manually defining different DQTPs to target the generation description instead of the generated RDF graph [26]. This means the RDF graph can be validated before any data is generated, as the generation description reflects how the RDF graph will be formed.

2.2.4. *High-level language*

These approaches use a terse high-level language specifically designed to describe constraints for validation [51]. These languages are independent of underlying technologies, and alternative implementation strategies can be devised. We first discuss initial high-level languages, after which we discuss high-level languages with wide adoption from the community: ShEx and SHACL.

Description Set Profiles (DSP) [57] define a set of constraints using Description Templates, targeted specifically to Dublin Core Application Profiles, and implemented using SPIN [12]. Other high-level languages to describe constraints include OSLC Resource Shapes [68] – part of IBM Resource Shapes – and RDF Data Descriptions [30]. Luzzu [22] uses a custom declarative constraint language (Luzzu Quality Metric Language, LQML). Any metric that can be expressed in a SPARQL query can be defined using LQML. Moreover, quality dimensions other than the intrinsic dimensions are also expressible using LQML. Luzzu supports basic metrics and custom JAVA code allowing users to implement custom metrics.

ShEx Shape Expressions (ShEx) [65, 66] is a structural schema language which can be used for RDF graph validation. The grammar of ShEx is inspired by Turtle and RelaxNG, its semantics are well-founded, and its complexity and expressiveness are formalized [10, 70]. ShEx provides an extension point to handle advanced constraints via Semantic Actions, which allows to evaluate a part of the validated RDF graph using a custom function.

SHACL The Shapes Constraint Language (SHACL) [47] is the W3C Recommendation for validating RDF graphs against a set of constraints [47]. The core of SHACL is independent of SPARQL, which promotes the development of new algorithms and approaches to validate RDF graphs [52]. The original specification does not include a denotational semantics such as ShEx, however, the recent work of Corman et al. propose a concise formal semantics for SHACL's core constraint components, and a way of handling recursion in combination with negation [19]. Advanced features of SHACL include SHACL Rules (to derive inferred triples from the validated RDF graph) and SHACL Functions (to evaluate a part of the validated RDF graph using a custom function) [49].

2.3. Validation reports

Validation reports handle identification of which data quality dimensions are assessed in general, and the representation of violations in particular.

To identify data quality dimensions, Radulvic et al. extended the Dataset Quality Ontology (daQ) [23] to include all data quality dimensions as identified by Zaveri et al. [74], leading to the Data Quality Vocabulary [67]. This allows the comparison of data quality dimension coverage of different frameworks.

The violations report itself allows to distribute and compare the violations found in an RDF graph, and can refer to the dimension specifications using aforementioned general vocabularies. For example, the Quality Problem Report Ontology assembles detailed quality reports for all data quality dimensions [22]. The Reasoning Violations Ontology (RVO) is used to represent integrity constraint violations [15], and Kontokostas et al [50] use the RDF Logging Ontology[5] (RLOG) to describe RDFUnit's violation results. Both ShEx and SHACL provide violation report descriptions, with means to specify the violating resources, using a ShapeMap [66] and a Focus node [47], respectively.

2.4. Constraint types

Hartmann *né* Bosch et. al identify eighty-one general *constraint types* [14]. These constraint types are an abstraction of specific constraints, independent of the constraint language used to describe them. A constraint type can be defined in different ways. For example, the *property domain* constraint type specifies that resources that use a specific property should be classified via a specific class, e.g., all resources using the `:birthdate` property that are not classified as a `:Person` are violating resources. Using RDFS [16], the property domain constraint type can be assessed by interpreting `rdfs:domain` as an integrity constraint. Using SHACL, this can be achieved by defining a `sh:property` with `sh:class` for a `sh:targetSubjectsOf` shape [47].

Moreover, Hartmann et al. provide a logical underpinning stating the requirements for a validation approach to support all constraint types [13]. For thirty-five out of eighty-one constraints types (43.2%), reasoning (up to OWL-DL expressiveness) can improve the validation: without reasoning, either too many or too few violations can be returned.

3. Comparative analysis

Different types of validation approaches are proposed in the state of the art. The most prominent approaches are *hard-coded*, based on *integrity constraints*, *query-based*, and using *high-level languages*. In this section, we compare them with our proposed *rule-based reasoning* approach. Our analysis is summarized in Table 1.

We adapt the framework presented by Pauwels et al. [62], which introduces comparative factors of key implementation strategies for compliance checking applications. We adjust these factors with respect to the validation problems identified in Section 1.1. We generalize the factors *time*, *customization*, and *inferencing steps*, and introduce *explanation* and *reasoning preprocessing* as validation-specific factors.

Explanation The explanation as to why a certain violation occurs (i.e., the root cause). The more specific a validator can explain, the easier it is to (automatically) refine the RDF graph and improve its quality. Existing approaches typically have the means to explain violations up to the level of which resource violates which constraint. Explanations of *hard-coded* approaches either need to be explicitly implemented, or are provided by inspecting the code base. When using *integrity constraints*, approaches exist for resolving inconsistencies. These approaches perform some sort of root cause analysis, but are usually targeted at refining the axioms of the ontologies themselves [33]. It is not a standard

[5]http://persistence.uni-leipzig.org/nlp2rdf/ontologies/rlog#

Table 1

Comparing the prominent validation approaches with rule-based reasoning, using factors *explanation*, *time*, *customization*, *inferencing steps*, and *reasoning preprocessing*. The *time* row indicates which approaches' execution time is influenced due to the reasoning preprocessing using an asterisk. The asterisk in the *inferencing steps* row indicates that approaches based on integrity constraints cannot combine with a custom set of inferencing steps that overlaps with the integrity constraints, as their semantics are redefined.

Approach	**Hard-coded**	**Integrity constraints**	**Query-based**	**High-level language**	***Rule-based reasoning***
Explanation	No	Limited / Yes	Limited	Limited	Yes
Time	Short*	Long	Long*	Short*	Long
Customization	Limited	Limited	Open	Open	Open
Inferencing steps	No / Limited	Yes*	Limited / Yes	Limited / Yes	Yes
Reasoning preprocessing	Yes	Limited	Yes	Yes	N/A

feature to produce proofs of the results of description logic-based reasoners [58]. In a *query-based* approach, the used SPARQL endpoint returns bindings [1]. In the case of validation, it returns the violating resources, without additional explanation. *High-level languages* can have mechanisms to additionally include the violating resources in the validation report. For example, ShEx and SHACL provide ShapeMaps [66] and Focus nodes [47], respectively. SHACL's Focus nodes can further specify which predicate and object cause the violation, except for, e.g., compound constraints. Using *rule-based reasoning* allows the generation of a logical proof, as rule-based reasoning relies on a general "implies" construct to describe rules, and rule-based reasoners typically do not contain description logic optimizations. Such a logical proof declares which rules were triggered to arrive at a certain conclusion, giving a precise explanation for the root causes of constraint violations. Where existing approaches typically have the means to explain violations up to the level of which resource violates which shape, a logical proof can provide a more detailed explanation.

Time The time needed to execute the validation: short versus long. Typically, specialized approaches allow for optimizations, making them faster than general approaches. *Hard-coded* is usually the fastest and needs the shortest processing time, followed by systems that use *high-level languages*: both can be optimized for validation tasks. The other approaches (using *integrity constraints*, *query-based*, and *rule-based reasoning*) are typically built using an underlying existing technology (description logic-based reasoners, SPARQL endpoints, and rule-based reasoners, respectively). They are not built (or optimized) for validation tasks. This makes them independent of the constraint language, but can also slow down the validation. The total execution time of validation approaches depends on whether a reasoning preprocessing step to include additional inferencing steps is required or not. Using rule-based reasoning is thus potentially slower than existing approaches, however, it does not require inclusion of reasoning preprocessing.

Customization The extent of customization each type of approach enables. Typically, ease of customization is improved by using a declarative language. Customization of a *hard-coded* system requires development effort, as the business logic is embedded within the code. Other approaches rely on declarations to customize the validation. Declarations are decoupled, i.e., independent of the tool's implementation. Thus, they can be shared and easier customized to a certain use case. Description logic-based reasoners used to identify *integrity constraints* are typically optimized for description logics such as OWL-QL and OWL-DL. Customization is limited to the description logic that the reasoner is optimized for. *Query-based* approaches allow customization by defining additional SPARQL queries and registering custom functions [34]. Systems using *high-level languages* are customized using the declarations as specified by the used language. The adoption of ShEx and SHACL shows that these languages provide sufficient customization. The extension mechanisms of these languages such as Semantic Actions [66] and SHACL Advanced Features [49], respectively, allow to customize the validation even further. Using *rule-based reasoning* allows customization by adding and removing rules. As opposed to existing approaches, users can customize both the constraint types and the set of inferencing steps within the same declarative language.

Inferencing steps Whether the validation approach supports a (custom) set of inferencing steps. *Hard-coded* systems can support a fixed set of inferencing steps, but this set cannot be inspected or altered without investigating the code base. Approaches that use *in-*

tegrity constraints for validation propose alternative semantics of commonly agreed upon ontology languages to include, a.o., some form of CWA [54, 71]. This leads to ambiguity in the Semantic Web as an existing, globally agreed upon logic, is changed [3]. It is not possible to combine such validation with a (custom) set of inferencing steps within a description logic: the same inferencing step has different semantics whether it is used for validation or for inferring new statements. SPARQL endpoints used for *query-based* approaches can support up to OWL-RL reasoning [46], or support up to RDF and RDFS entailment via translation of the SPARQL queries using property paths [71]. *High-level languages* such as SHACL allow specifying the used entailment regime [32]: SHACL validators may operate on RDF graphs that include entailments using the `sh:entailment` property [47]. Furthermore, SHACL Rules [49] can be used to a certain extent to generate inferred statements during validation. By design, *rule-based reasoning* allows inclusion of a set of additional (custom)) inferencing rules [59]. Whereas existing approaches mostly allow configuration to support, e.g., a specific entailment regime, the customization of the set of inferencing steps is more fine-grained for rule-based reasoners. This can increase complexity, but also allows catering the validation to use cases that depend on a specific set of inferencing steps. The importance of such use cases is evidenced by the fact that SHACL Rules is proposed as an advanced feature to the SHACL specification [49].

Reasoning preprocessing Existing approaches have no support for including a custom set of inferencing steps, propose alternative semantics, or allow a specific entailment regime. By including a reasoning step as preprocessing step to these approaches (see Fig. 1.1), the entailments valid during validation can be matched with the entailments valid for the use case, even when that use cases requires a custom set of inferencing steps [13]. First, a reasoner – optionally, hence the dashed line – infers all valid entailments of the original RDF graph (Fig. 1.1, *Reasoner*), taking into account the axioms of the relevant ontologies and vocabularies (*Axioms*). Then, the newly generated RDF graph (*RDF graph**) is validated with respect to the specified constraints (Fig. 1.1, *Validator*).

By using a preprocessed inferred RDF graph, multiple systems (i.e., the reasoner and the validator) need to be combined, configured, and maintained. This separates concerns, however, this also means that different languages may need to be learned and combined for specifying the inferencing steps and constraints. As these multiple systems are not aligned, the reasoner could infer a large number of new triples that are irrelevant to the defined constraints, which could lead to bad scaling (Fig. 1.1, *RDF graph**). Also, explaining the violation is hindered. Even when the reasoner can differentiate between the original triples and the inferred triples, finding the root causes involves investigating the output of both systems: the validator detecting the violations, and the reasoner inferring the supported entailments.

Reasoning preprocessing is not required when using *rule-based reasoning*. The set of inferencing steps and the set of constraints can be defined using the same declaration (Fig. 1.2, *Inferencing rules* and *Constraints**), and executed simultaneously on the RDF graph and the axioms. Which statements need to be inferred can be optimized guided by the set of constraints, and only the output of a single system needs to be investigated to explain the found violations.

4. Logical Requirements

In this section, we discuss the requirements for a logic to be a valid choice for RDF graph validation, and argue for using a rule-based logic.

Constraint languages need to cope with different constraint types depending on users' needs. Each constraint type implies logical requirements. The constraint types and the requirements they entail are investigated by Hartmann et al., claiming that Closed World Assumption (CWA) and Unique Name Assumption (UNA) are crucial for validation [13]. These requirements are not common for Semantic Web logics, as data on the Web is decentralized, information is spread ("anyone can say anything about anything" [20]), and single resources can have multiple URIs. Instead, Semantic Web logics such as OWL-DL assume OWA and in general non-Unique Name Assumption [55]. Hartmann et al. emphasize the difference between reasoning and validation, and favor query-based approaches for validation. The latter – when needed – can be combined with, e.g., OWL-DL or OWL-QL reasoning as a preprocessing step.

However, we show that Semantic Web rule-based reasoning *can* be used for validation, even though typically CWA and UNA is not followed. Specifically, we state that the requirements for using rule-based reasoning are (i) supporting Scoped Negation as Failure (SNAF) [21, 45, 64] instead of CWA (Section 4.1),

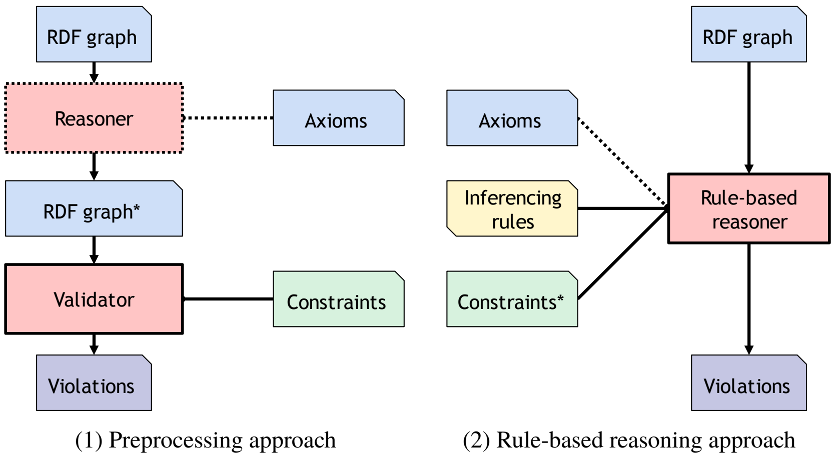

Figure 1. The preprocessing approach: first (optionally, hence the dashed line), a *reasoner* is used to generate intermediate data (*RDF graph**). That intermediate data is then the input data for the *Validator*. Using a rule-based reasoner only needs a single system and language to combine reasoning and validation.

(ii) containing predicates to compare URIs and literals instead of supporting UNA (Section 4.2), and (iii) supporting expressive built-ins, as validation often deals with, e.g., string comparison and mathematical calculations (Section 4.3).

4.1. Scoped Negation as Failure

Existing works claim that CWA is needed to perform validation [13, 60, 71]. Given that most Web logics assume OWA, this would require semantic redefinitions to include inferencing during validation [54], which leads to ambiguity. However, as validation copes with the *local knowledge base*, and not the entire Web, we claim Scoped Negation as Failure (SNAF) is sufficient. This is an interpretation of logical negation: instead of stating that ρ does not hold (i.e., $\neg\rho$), it is stated that reasoning fails to infer ρ within a specific *scope* [21, 45, 64]. This scope needs to be explicitly stated. As such, SNAF keeps monotonicity.

To understand the idea behind Scoped Negation as Failure, let us validate following RDF graph:

$$\texttt{:Kurt a :Researcher;} \tag{4}$$

$$\texttt{:name "Kurt01".} \tag{5}$$

We validate the constraint "every individual which is declared as a researcher is also declared as a person". This thus means a violation is returned when an individual is found during validation which is a researcher, but not a person:

$$\begin{aligned}\forall x : ((\ \texttt{x a :Researcher}\ )\wedge\\ \neg(\ \texttt{x a :Person}\ ))\\ \rightarrow (\ \texttt{:constraint :isViolated "true".}\ )\end{aligned} \tag{6}$$

As stated, this constraint cannot be tested with OWA: the knowledge base contains the triple of formula (4), but not of:

$$\texttt{:Kurt a :Person.} \tag{7}$$

The rule is more general: given its open nature, we cannot guarantee that there is no document in the entire Web which declares the triple of formula (7).

This changes if we take into account SNAF. Suppose that $\mathcal{K}$ is the set of triples we can derive (either with or without reasoning) from our knowledge base of formulas (4) and (5). Having $\mathcal{K}$ at our disposal, we can test:

$$\begin{aligned}\forall x : (((\ \texttt{x a :Researcher}\ ) \in \mathcal{K})\wedge\\ \neg((\ \texttt{x a :Person}\ ) \in \mathcal{K}))\\ \rightarrow (\ \texttt{:constraint :is :violated.}\ )\end{aligned} \tag{8}$$

The second conjunct is not a simple negation, it is a negation with a certain scope, in this case $\mathcal{K}$. If we add new data to our knowledge base, e.g., the triple of formula (7), we would have a different knowledge base $\mathcal{K}'$ for which other statements hold. The truth value of formula (8) would not change since this formula explicitly mentions $\mathcal{K}$. SNAF is what we actually need for validation: we do not validate the Web in general, we validate a specific RDF graph.

4.2. Predicates for Name Comparison

UNA is deemed required for validation [13], i.e., every resource taken into account can only have one single name (a single URI in our case) [44]. UNA is in general difficult to obtain for the Semantic Web and Web logics due to its distributed nature: different RDF graphs 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 Britain as, e.g., `dbpedia-nl:London`. That fact is even stated in the corresponding datasets using the predicate `owl:sameAs`. The usage of `owl:sameAs` conflicts with UNA and influences validation [13].

Let us look into the following example. We assume `dbo:capital` is an `owl:InverseFunctionalProperty`. Our knowledge base contains:

$$\texttt{:Britain dbo:capital :London.} \tag{9}$$

$$\texttt{:England dbo:capital :London.} \tag{10}$$

Since both `:Britain` and `:England` have `:London` as their capital and `dbo:capital` is an inverse functional property, an description logic-based reasoner would derive that

$$\texttt{:Britain owl:sameAs :England.} \tag{11}$$

This thus influences the validation result. Such a derivation cannot be made if UNA holds, since UNA explicitly excludes this possibility.

The related constraint – defined as `INVFUNC` by Kontokostas et al. [50] – specifies that each resource should contain exactly one relationship via `dbo:capital`, i.e., the capital is different for every resource. The constraint `INVFUNC` is related to `owl:InverseFunctionalProperty`, but it is slightly different: while OWL's inverse functional property refers to *the resources* that are in the domain of `dbo:capital`, the validation constraint `INVFUNC` refers to *the representation of those resources*. The RDF graph of formulas (9) and (10) thus violates the `INVFUNC` constraint. Even if our logic does not follow UNA, this violation can be detected if the logic offers predicates to compare the (string) representation of resources.

4.3. Expressive Built-ins

Validation often deals with, e.g., string comparison and mathematic calculations. These functionalities are widely spread in rule-based logics using *built-in functions*. While it typically 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), whose aim is to provide a formalism to exchange rules in the Web [43]. 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.

Let us take a closer look to the comparison of URIs from the previous section. `func:compare` can be used to compare two strings. 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://example.com/Britain"
 "http://example.com/England")
                    func:compare -1.
```
(12)

To refer to a URI value, RIF provides the predicate `pred:iri-string` which converts a URI to a string and vice versa. To enable a rule to detect whether the two URI names are equal or not, an additional function is needed: the reasoner has to detect whether the comparison's result is different from zero. That can be checked using the predicate `pred:numeric-not-equal`, which is the RIF version of $\neq$ for numerical values. In the example, the comparison would be `true` since $0 \neq -1$. Using these RIF built-ins, a reasoner can check the name equality between `:Britain` and `:England`, and return a violation. Whether a rule based Web logic is suited for 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. Application

In this section, we present our approach that uses rule-based reasoning for validation. We discuss the different components and the workflow in Section 5.1, the underlying technologies in Section 5.2, and implementation in Section 5.3. We end with an example using rules in Section 5.4.

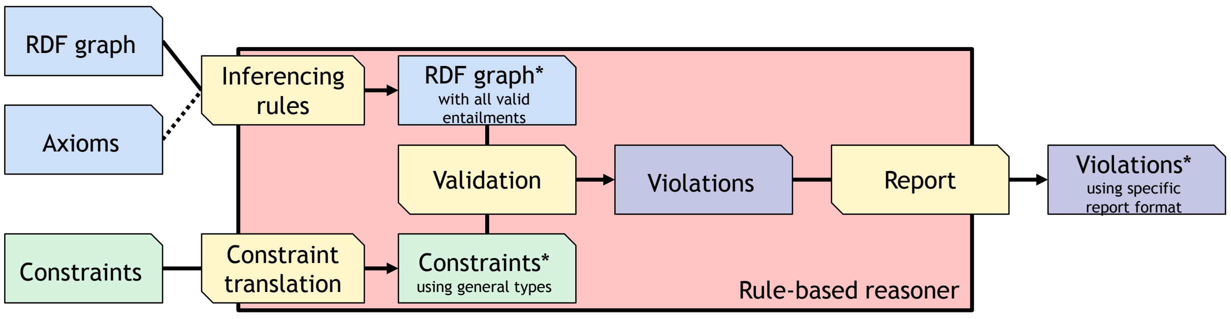

Figure 2. Components view of our approach. All double-snipped rectangles are rule sets, the single-snipped rectangles are RDF graphs or constraint declarations. The large overlapping rectangle is the rule-based reasoner. By taking all rule sets into account, the rule-based validator is formed. Four parts can be identified within the validation execution: (i) possibly guided by provided *Axioms*, all supported entailments of the given *RDF graph* can be generated using the *Inferencing rules*, resulting in *RDF graph**; (ii) the general *Constraints** are inferred from the given *Constraints* using a set of rules for *Constraint translation*; (iii) the rules for *Validation* generate *Violations*; and (iv) the returned *Violations** are structured given a set of rules that specify the *Report* format.

5.1. Customizable validation

Our validator consists of multiple components that can be configured by adjusting the different rule sets (Fig. 2). The execution is primarily handled using the rule-based reasoner as underlying technology.

The set of *Inferencing rules* specifies the supported entailments during validation. This set can either be a predefined set to support, e.g., RDFS entailment [16], or can be fully customized. Optionally, the relevant axioms are provided during validation. As such, the entailments supported by the use case can be matched during validation.

The set of rules forming the *Constraint translation* allows our validator to infer the general constraint types – common across existing constraint languages [14] – from specific constraint descriptions. It can thus infer these types from the constraints described in a specific language such as SHACL [47]. The general constraint types are described using RDF-CV[6] [11], which generalizes the constraint types into a coherent structure. Our rule-based validator is thus constraint language-independent.

The set of rules forming the *Validation* allows our validator to infer violations on the RDF graph with all supported entailments, based on the general constraint types. This set of rules specifies how to detect each constraint type.

The set of rules forming the *Report* allows our validator to infer the resulting violations in the required format. This set can be adapted to, e.g., the SHACL report format [47].

As a result, this declarative approach is decoupled from ontology language, constraint language, and report format. When no additional rule sets are included (i.e., only the *Validation* rule set is used), this validator does not infer any entailments, only validates constraints described using RDF-CV, and returns a report in a format based on RDF-CV.

All rule sets and input data are taken into account during a single reasoner execution. As opposed to using a reasoning preprocessing step, the inferred entailments can be geared towards the specified constraints (when making use of a backward chaining reasoner), and no unnecessary entailments are produced. For example, when an axiom specifies the range of a certain path, but no constraints are related to that path, this range might not need to be inferred. Moreover, as you only have a single system, finding the root cause does not require investigation of multiple systems: the logical proof contains the complete overview of which rules were used to generate which entailments and which violations.

5.2. Technologies

The most important technological considerations are the rule-based web logic and reasoner in accordance with that logic. Rule-based web logics include the Semantic Web Rule Language (SWRL) [42] and N3Logic [8]. However, as opposed to N3Logic, SWRL does not support SNAF[7], a logical requirement for being used for validation. N3Logic supports at least

[6]https://github.com/boschthomas/RDF-Constraints-Vocabulary

[7]https://github.com/protegeproject/swrlapi/wiki/SWRLLanguageFAQ#Does_SWRL_support_Negation_As_Failure

OWL-RL inferencing [2], which can be included during validation.

The *rule language* introduced together with N3Logic is N3 [7]. Everything covered by RDF 1.1 Semantics [37] is covered in N3. Syntactically, it is a superset of Turtle [4]. N3 allows declaring inferencing rules, axioms, and constraints in the same language. As in RDF, blank nodes are understood as existentially quantified variables and the co-occurrence of two triples as in the RDF graph of formulas (9) and (10) is understood as their conjunction. More, N3 supports universally quantified variables, indicated by a leading question mark `?`.

```
?x :likes :IceCream.
```
(13)

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

$$\forall x : \text{likes}(x, \text{ice-cream}) \tag{14}$$

Rules are written using curly brackets `{ }` and the implication symbol `=>`. An `rdfs:subClassOf` relation such as `:Person rdfs:subClassOf :Researcher` can be expressed as:

```
{?x a :Researcher} => {?x a :Person}.
```
(15)

The general `rdfs:subClassOf` relation can be expressed as:

```
{?C rdfs:subClassOf ?D. ?X a ?C}
        => {?X a ?D}.
```
(16)

Reasoners that support N3Logic include FuXi, cwm, and EYE. FuXi[8] is a forward chaining production system for N3 whose reasoning is based on the Rete algorithm [31]. The forward chaining cwm [6] reasoner is a general-purpose data processing tool which can be used for querying, checking, transforming and altering information. EYE[9] [73] is a high-performance reasoner written in Prolog, enhanced with Euler path detection, allowing the creator of the rules to decide when to do forward reasoning and when backwards. EYE has generous support for built-in functions[10], among which, the RIF functions. We choose the EYE reasoner as it fulfills the requirements as presented in Section 4. Furthermore, its ability to combine forward and backward chaining proves especially useful since constraint types are mostly localized to single relationships [13]. This means backward chaining has a potentially large impact on the performance: reasoning during validation can be very targeted, and in most cases, only facts that are relevant to the defined constraints are inferred.

5.3. Implementation

Our implementation is dubbed "Validatrr": a validator using rule-based reasoning. A Node.js JavaScript framework was created to discover and retrieve the vocabularies and ontologies as required by the use case, manage the commandline arguments, etc. The implementation is available at https://github.com/IDLabResearch/validatrr, and the set of validation rules (Fig. 2, center) is available at https://github.com/IDLabResearch/data-validation.

5.4. Execution example

As example, we validate an RDF graph with a custom set of inferencing steps using SHACL constraints. We take into account the example of the introduction (formula (1)), but the case where `:Bob` has two birthdates defined. The implications of `rdfs:domain` (formula (2)) should be taken into account as defined in RDFS [16] during validation, and the SHACL constraint states that each person should have exactly one birthdate (Listing 1). The result should be in the SHACL validation report format. Using this example, we can detail every step as show in Fig. 2: the RDF graph with all supported entailments (*RDF graph**) and general constraint types (*Constraints**) are inferred using a (custom) set of inferencing rules (*Inferencing rules*) and constraint translation rules (*Constraint translation*), after which the validation occurs (*Validation*), and the resulting violations are translated via rules (*Report*) in a specific report format (*Violations**).

```
:PersonShape a sh:NodeShape ;
  sh:targetClass :Person ;
  sh:property [
    sh:path :birthdate ;
    sh:minCount 1 ; sh:maxCount 1 ;
    sh:datatype xsd:date ] .
```

Listing 1: Person Shape in SHACL

[8] http://code.google.com/p/fuxi/
[9] https://github.com/josd/eye
[10] http://eulersharp.sourceforge.net/2003/03swap/eye-builtins.html

To make sure `rdfs:domain` is correctly interpreted during validation, we include additional inferencing rules[11] (*Inferencing rules*), described in N3 as

```
{?P rdfs:domain ?C. ?X ?P ?Y}
          => {?X a ?C} .
```
(17)

Given formula (17), it is inferred that `:Bob` is a person (*RDF graph**).

To make sure SHACL constraints are correctly interpreted, SHACL translation rules need to be included during validation (*Constraint translation*). The general "Exact Qualified Cardinality Restrictions" RDF-CV constraint is inferred from the SHACL constraint of Listing 1, using the rules of Listing 2 (*Constraints**).

```
{
  ?sh a sh:NodeShape ;
    sh:targetClass ?Class ;
    sh:property [
      sh:path ?p ;
      sh:minCount ?v ; sh:maxCount ?v1 ;
    sh:datatype ?C ] .
  ?v pred:numeric-equal ?v1
} => {
  ?constraint a rdfcv:SimpleConstraint ;
    :originalShape ?sh ;
    :constraintType :ExQualCardRestr ;
    rdfcv:constrainingElement
      :exact-cardinality ;
    rdfcv:contextClass ?Class ;
    rdfcv:leftProperties ?p ;
    rdfcv:classes ?C ;
    rdfcv:constrainingValue ?v
} .
```

Listing 2: Translate the SHACL shape to a general constraint type

Validation makes use of general rules, i.e., Listing 3 (*Validation*). Lines 11–14 define how to find a violation, relying on built-ins: gather a set of resources in a list (`e:findall`), calculate the length of that list (`e:length`), and mathematically compare numbers (`math:notEqualTo`). For all objects of a certain class or datatype related using predicate `?p` (in this case `:birthdate`) where the number of objects is different from the constraint value `?v` (in this case 1), a violation is returned (lines 16–21).

[11] http://eulersharp.sourceforge.net/2003/03swap/rdfs-domain.html

```
1  {
2    ?constraint a rdfcv:SimpleConstraint ;
3      :constraintType :ExQualCardRestr ;
4      rdfcv:constrainingElement
5        :exact-cardinality ;
6      rdfcv:contextClass ?Class ;
7      rdfcv:leftProperties ?p ;
8      rdfcv:classes ?C ;
9      rdfcv:constrainingValue ?v .
10   ?x a ?Class.
11   _:x e:findall
12     ( ?C {?x ?p ?o. ?o a ?C} ?list) .
13   ?list e:length ?l .
14   ?l math:notEqualTo ?v
15 } => {
16   _:v a :constraintViolation ;
17     :violatedConstraint ?constraint ;
18     :class ?Class ;
19     :instance ?x ;
20     :objectClass ?C ;
21     :property ?p
22 } .
```

Listing 3: Validate using general constraint types

The general violations are translated into a report format (Fig. 2, *Violations**), e.g., using the SHACL Validation Report [47] (see Listing 4). The result is a set of triples using the exact same input and output as a SHACL processor. However, the RDF graph's supported entailments can be matched to the use case, and the process is a single reasoning execution with transparent rule sets.

```
{
  _:v a :constraintViolation ;
    :violatedConstraint [
      :originalShape ?sh ;
      :constraintType :exact-cardinality
    ] ;
    :class ?Class ;
    :instance ?x ;
    :objectClass ?C ;
    :property ?p
} => {
  _:y a sh:ValidationReport ;
    sh:conforms false ;
    sh:result [
      a sh:ValidationResult ;
      sh:resultSeverity sh:Violation ;
      sh:focusNode ?x ;
      sh:resultPath ?p ;
      sh:resultMessage "No exact match" ;
      sh:sourceShape ?sh ]
} .
```

Listing 4: Translate the general violations to the SHACL validation report

Moreover, different constraint descriptions are easily supported via the general constraint types. Given the OWL restriction of Listing 5: using a different set of rules, we can translate this restriction into the same constraint type (Listing 6). The validation process continues exactly the same.

```
:Person rdfs:subClassOf _:x .
_:x a owl:Restriction ;
  owl:onProperty :birthdate ;
  owl:qualifiedCardinality
      "1"^^xsd:nonNegativeInteger ;
  owl:onDataRange xsd:date .
```

Listing 5: An OWL restriction

```
{
  ?Class rdfs:subClassOf ?c .
  ?c a owl:Restriction ;
    owl:onProperty ?x ;
    owl:qualifiedCardinality ?v ;
    owl:onDataRange ?C
} => {
  ?constraint a rdfcv:SimpleConstraint ;
    :originalShape _:x ;
    :constraintType :ExQualCardRestr ;
    rdfcv:constrainingElement
      :exact-cardinality ;
    rdfcv:contextClass ?Class ;
    rdfcv:leftProperties ?p ;
    rdfcv:classes ?C ;
    rdfcv:constrainingValue ?v
} .
```

Listing 6: Translate the OWL restriction to the general constraint type

6. Hypothesis validation

To validate the hypotheses of Section 1.2, we compare Validatrr to different validation approaches. We show that Validatrr (i) accurately explains the root cause of why a violation occurs in more cases than specified in SHACL, given the SHACL core constraint components (accepting Hypothesis 1, see Section 6.1); (ii) returns an accurate number of validation results with respect to the used set of inferencing steps, compared to an integrity constraints validator with a fixed set of inferencing steps using RDFUnit (accepting Hypothesis 2, see Section 6.2); and (iii) supports an equivalent number of constraint types than existing approaches (accepting Hypothesis 3, see Section 6.3). The performance evaluation shows that our implementation is faster than the state of the art when combining inferencing and validation for commonly published datasets (accepting Hypothesis 4, see Section 6.4).

6.1. Root cause explanation of constraint violations

Using the logical proof, we increase the explanation's accuracy compared to what is currently expected of a validation approach. SHACL is a W3C Recommendation standardizing the description of constraints and violation reports for RDF graph validation. We show that the logical proof produced by the rule-based reasoning execution provides more detailed root cause explanations of constraint violations, compared to SHACL's violation report description.

The SHACL recommendation provides a set of test cases, enabling implementations prove compliance[12]. The validation report denotes the violating resources via `sh:focusNode`, and in some cases can further specify the violating path via `sh:resultPath` and the violating value via `sh:value` [47]. However, it is not always possible to retrieve such additional information about the root cause. We revisit the previous example constraint that given a resource r, this resource has $(r_{firstname} \wedge r_{lastname}) \vee (r_{nickname})$[13]. Validation of formula (1) using a conform SHACL implementation results in a validation report similar to Listing 7. The validation report does not provide any further details to explain why `:Bob` is invalid[14].

```
[ rdf:type sh:ValidationReport ;
  sh:conforms "false"^^xsd:boolean ;
  sh:result [
    rdf:type sh:ValidationResult ;
    sh:focusNode :Bob ;
    sh:resultSeverity sh:Violation ;
    sh:sourceConstraintComponent
```

[12]https://github.com/w3c/data-shapes/tree/gh-pages/data-shapes-test-suite/tests

[13]This example is similar to the following SHACL test case: https://github.com/w3c/data-shapes/blob/gh-pages/data-shapes-test-suite/tests/core/node/or-001.ttl

[14]https://www.w3.org/TR/shacl/#validator-OrConstraintComponent

```
        sh:OrConstraintComponent ;
      sh:sourceShape :PersonNameShape ;
      sh:value :Bob ; ] ; ]
```

Listing 7: Validation report of an OR constraint

The rule-based reasoning execution of Validatrr can generate a proof, showing the rules used to reach a conclusion. This logical proof allows to determine, for each violation, which part of the RDF graph is the root cause of the violation, and which axiom of the used ontology triggered an inference causing the violation. Listing 8 shows the part of the proof which contains the rules deriving the violation. For `:firstname`, `:lastname`, and `:nickname`, we query objects that are linked using the respective predicate (Listing 8, lines 12–15, 18–21, and 24–27). $\mathcal{K}$ is the scope of our knowledge base, in which we look for violations. We count the number of objects found and compare them with the needed number. For `:firstname`, one linked object is found (Listing 8, lines 16–17), however, no linked object is found for `:lastname` nor `:nickname` (Listing 8, lines 22–23 and 28–29): a violation is returned.

```
 1  <#lemma20> a r:Inference;
 2  r:gives {
 3  _:b1 a :constraintViolation.
 4  _:b1 :violatedConstraint _:b2.
 5  _:b1 :class :Man.
 6  _:b1 :instance :Bob.
 7  _:b1 :property :lastname.
 8  _:b1 :property :nickname. };
 9  r:evidence (
10  ...
11  <#lemma37>
12  [ a r:Fact; r:gives { (𝒦 1) e:findall
13  (1
14  {:Bob :firstname _:b3}
15  (1))}]
16  [ a r:Fact; r:gives {(1) e:length 1}]
17  [ a r:Fact; r:gives {1 math:greaterThan 0}]
18  [ a r:Fact; r:gives {(𝒦 1) e:findall
19  (1
20  {:Bob :lastname _:b3}
21  ())}]
22  [ a r:Fact; r:gives {() e:length 0}]
23  [ a r:Fact; r:gives {0 math:lessThan 1}]
24  [ a r:Fact; r:gives {(𝒦 1) e:findall
25  (1
26  {:Bob :nickname _:b3}
27  ())}]
28  [ a r:Fact; r:gives {() e:length 0}]
29  [ a r:Fact; r:gives {0 math:lessThan 1}]).
```

Listing 8: Validation proof of an OR constraint

Due to this proof, Validatrr can provide detailed explanations for the root causes of violations for all SHACL core constraint components, compared to 46%–75% of SHACL-conforming implementations. Analysis of the SHACL specification shows that, out of the 28 core constraint components, 13 (46%) provide a full explanation of the root cause (summarized in Table 2). For eight of the remaining components (an additional 29%), the validation report returns which resource violates which constraint, but does not return a detailed explanation. For example, a `sh:class` violation occurs when the targeted node is a literal, *or* when the targeted node is not classified accordingly, but this disjunction is not reflected in the validation report. For the remaining seven components, the validation report does not provide an explanation at all. For example, violations of nested shapes are not reflected in the validation report, only violations of top-level shapes.

Compared to SHACL-conforming implementations, Validatrr supports, a.o., explanation of disjunction and nested shapes. Our approach provides detailed explanations for all core components of W3C's recommended high-level language to describe constraints. We thus accept Hypothesis 1.

6.2. Accurate number of found violations

Validatrr finds a more accurate number of violations compared to the state of the art. To prove this, we first compare Validatrr with the state of the art functionally, and then include a set of inferencing steps to clarify the difference.

Specifically, we compare with RDFUnit [50]. Hartmann et. al explicitly proposed using query-based approaches for validation [11], and RDFUnit is such a query-based approach, relying on a SPARQL endpoint, and describing the constrains using SPARQL templates named Data Quality Test Patterns (DQTP). As such, RDFUnit is highly configurable and one of the implementations that supports SHACL[15].

Functional comparison We compare with the original pattern library of RDFUnit [50]. This pattern library is the closest to the constraint types as introduced by Hartmann et al. [14]: the mapping between those two is presented in previous work [3]. We test all unit tests defined by RDFUnit[16] after retrieving them as-is from the RDFUnit repository. As Validatrr validates

[15]https://w3c.github.io/data-shapes/data-shapes-test-suite/

[16]https://github.com/AKSW/RDFUnit/tree/master/rdfunit-core/src/test/resources/org/aksw/rdfunit/validate/data

Table 2

Analysis of root cause explanation of violations for SHACL core constraint components. Validatrr can provide more detailed explanations for up to 56% of the components compared to SHACL-conforming implementations.

SHACL Name	Root Cause Explanation	Comment
`sh:class`	~	disjunction
`sh:datatype`	~	disjunction
`sh:nodeKind`	~	disjunction
`sh:minCount`	✗	no explanation
`sh:maxCount`	✗	no explanation
`sh:minExclusive`	✓	
`sh:minInclusive`	✓	
`sh:maxExclusive`	✓	
`sh:maxInclusive`	✓	
`sh:minLength`	~	disjunction
`sh:maxLength`	~	disjunction
`sh:pattern`	✓	
`sh:languageIn`	~	disjunction
`sh:uniqueLang`	✗	no explanation
`sh:equals`	✓	
`sh:disjoint`	✓	
`sh:lessThan`	✓	
`sh:lessThanOrEquals`	✓	
`sh:not`	✓	
`sh:and`	~	conjunction
`sh:or`	~	disjunction
`sh:xone`	✓	
`sh:node`	✗	nesting
`sh:property`	✗	nesting
`sh:qualifiedValueShape`, `sh:qualifiedMinCount`, `sh:qualifiedMaxCount`	✗	nesting
`sh:close`, `sh:ignoredProperties`	✓	
`sh:hasValue`	✓	
`sh:in`	✗	nesting

general constraint types, a custom profile was created that translates the RDFUnit patterns to general constraint types. For a detailed explanation of the different test cases, we refer to the original RDFUnit paper [50].

The validation results depend on the used set of inferencing steps. RDFUnit implicitly takes "every resource is an `rdfs:Resource`" and the `rdfs:subClassOf` construct into account, forming the custom set of inferencing steps υ. We compare RDFUnit with Validatrr using three sets of inferencing steps, taking into account (i) no entailment at all ($\emptyset$), (ii) the custom set of inferencing steps (υ), and (iii) full RDFS entailment (ρ).

Table 3

Comparing RDFUnit to Validatrr using different sets of inferencing steps ($\emptyset$, υ, and ρ). Validatrr finds more violations given the same set of inferencing steps, and the set of inferencing steps used impacts the result. Test cases where Validatrr outperforms RDFUnit are starred. Rows where Validatrr and RDFUnit differ are marked gray.

Test Case	**# found violations**			
	RDFUnit	Validatrr		
	υ	$\emptyset$	υ	ρ
INVFUNC_correct	0	0	0	0
INVFUNC_wrong	2	0	2	2
OWLCARDT_correct	0	0	0	0
OWLCARDT_wrong_exact	6	6	6	6
OWLCARDT_wrong_max	2	2	2	2
OWLCARDT_wrong_min	2	2	2	2
OWLDISJC_correct	0	0	0	2
OWLDISJC_wrong	6	2	6	6
OWLQCARDT_correct	0	0	0	0
OWLQCARDT_wrong_exact	6	6	6	6
OWLQCARDT_wrong_max	2	2	2	2
OWLQCARDT_wrong_min	2	2	2	2
RDFLANGSTRING_correct	0	0	0	0
RDFLANGSTRING_wrong	2	2	2	0
*RDFSRANGE-MISS_wrong**	1	3	3	0
RDFSRANGED_correct	0	0	0	0
*RDFSRANGED_wrong**	2	3	3	0
*RDFSRANGE_correct**	0	5	4	0
*RDFSRANGE_wrong**	1	3	3	3
RDFSRANG_LIT_correct	0	0	0	0
RDFSRANG_LIT_wrong	3	3	3	1

Table 3 summarizes the results. For each constraint, we mention the test case's name, the number of violations that RDFUnit detects, and the number of violations that Validatrr detects using the different sets of inferencing steps. The table shows the impact of using different sets of inferencing steps: depending on the set, Validatrr finds a different number of violations. More, Validatrr detects more violations using the same set of inferencing steps: there is a higher number of found violations for Validatrr under υ compared to RDFUnit.

Validatrr finds more violations and supports more constraint types than RDFUnit, denoted as starred test cases RDFSRANGE-MISS_wrong, RDFSRANGED_wrong, RDFSRANGE_correct, and RDFSRANGE_wrong. RDFUnit does not yet support the constraint type *multiple ranges*: when a certain predicate is used, each resource linked as an object to that predicate should be classified into multiple classes. In all other cases, both solutions identify the same number of violations when using the same set of inferencing steps. Validatrr thus functionally outperforms the pattern library (i.e., the corresponding constraint types) of RDFUnit.

Impact of including sets of inferencing steps during validation Running Validatrr using different sets of inferencing steps impacts the number of found violations. Validatrr is designed to easily configure this set using inferencing rules (Fig. 2, top-left). The results are found in Table 3, comparing the different Validatrr columns. On the one hand, certain violations are not found without entailment ($\emptyset$), as is the case for INVFUNC_wrong and OWLDISJC_wrong. On the other hand, violations are resolved early-on when including RDFS entailment (ρ), as is the case for RDFLANGSTRING_wrong.

Compared to existing validation approaches, our approach allows including custom sets of inferencing steps during validation. The inferencing provenance is retained in the proof, as all inferencing occurs during a single reasoning execution. The logical proof can thus distinguish between violations that are caused due to constraint violations in the original RDF graph, or due to entailment during validation. We thus accept Hypothesis 2.

6.3. Equivalent number of constraint types

Validatrr can support an equivalent number of constraint types compared to existing validation approaches such as RDFUnit and SHACL. In the previous section, we showed we functionally outperform the original pattern library of RDFUnit whilst including a custom set of inferencing steps during validation. In this section, we compare our number of supported constraint types to that of SHACL [47].

We test Validatrr against general constraint types [36], to show that the number of supported constraint types is equivalent to SHACL. We do not test specifically against SHACL's test cases, as Validatrr is independent of the constraint language. On https://github.com/IDLabResearch/data-validation, we provide a set of test cases, used to test these different constraint types.

Hartmann et al. investigated the constraint type support of SHACL, and stated that its coverage is 52% [36]. We updated the coverage report as presented by Hartmann et al. to take the latest SHACL specification and advanced features into account [47, 49]. The relevant data is available at Appendix A and online at https://github.com/IDLabResearch/constraint-types-coverage. This updated report shows that SHACL's constraint type coverage is 84%.

Validatrr can cover up to 94% of all constraint types – given the current expressive support for built-ins – and has been tested to cover a similar number of constraint types as SHACL[17]. After including the rules for the remaining constraint types, we support an equivalent number of constraint types compared to SHACL. We thus accept Hypothesis 3.

Achieving 100% coverage (i.e., the remaining five constraint types) requires additional development on the reasoner to support specific built-ins. "Whitespace Handling" and "HTML Handling" require parsing built-ins, and "Valid Identifiers" requires a built-in to test URIs' dereferencability. The remaining two types ("Structure" and "Data Model Consistency") are general constraint types, defined by Hartmann et al., requiring SPARQL support. Supporting these constraint types requires a translation from SPARQL queries to N3 rules, for which we refer to related work [69].

6.4. Speed

A validation approach that supports a custom set of inferencing steps is faster than a validation system that includes a reasoning preprocessing step. We first compare the performance of Validatrr to that of RDFUnit, both without and with a custom set of inferencing steps.

For these performance evaluations, we used 300 data sets with sizes ranging from ten to one million triples, and an executing machine consisting of 24 cores (Intel Xeon CPU E5-2620 v3 @ 2.40GHz) and 128GB RAM. All evaluations were performed using untampered docker images for both approaches to maintain reproducibility, the different tests were orchestrated using custom scripts. All timings include the docker images' initialization time. The data is available at https://github.com/IDLabResearch/validation-benchmark/tree/master/data/validation-journal.

Performance comparison We compare the execution time of Validatrr to RDFUnit, following RDFUnit's original evaluation method. We use a default set of constraints for a fixed set of schemas, as defined by Kontokostas et al. [50]. We consider six commonly used schemas: FOAF, GeoSPARQL, OWL, DC terms, SKOS, and Prov-O. For each schema, we use RDF graphs of varying size. The validated RDF graphs' size range from ten triples to one million triples, in logarithmic steps of base ten. At most ten different RDF graphs

[17]The test report is available at https://github.com/IDLabResearch/validatrr/blob/v0.2.0/reports/validatrr-rdfcv-earl.ttl

– per schema, per RDF graph size – were downloaded, by querying LODLaundromat's SPARQL endpoint [5].

We validate the different RDF graphs against their respective schema using the default set of constraints and set of inferencing steps (υ) of RDFUnit, and measure total execution time of Validatrr and RDFUnit. The median execution time across all schemas is plotted against RDF graph size per approach in a log-log scale (see Fig. 3). To make sure we can combine execution times across schemas, we tested the null hypothesis that no significant difference in execution time was found between schemas, by performing an ANOVA statistical test with single factor "used schema" for measurement variable "execution time per triple", executed pairwise for all used schemas. The null hypothesis with $\alpha = 0.05$ was accepted for every pair. The number of found violations are not plotted, as statistical analysis shows no large correlation between execution time and number of found violations, neither for Validatrr or RDFUnit (-0.0203 and 0.0458, respectively).

Validatrr's execution time is highly correlated with the number of triples of the validated RDF graph. Regression analysis shows an R square value of 0.9998, the null hypothesis with $\alpha = 0.05$ is accepted: Validatrr's execution time grows linearly with respect to the size of the validated RDF graph. Meanwhile, the execution time of RDFUnit remains constant at around 30s. This could largely be due to the set-up time required by RDFUnit, however, the timings attained via RDFUnit's docker image does not allow us to draw further conclusions. The set-up time of RDFUnit thus possibly dominates the total execution time.

Without customizing the set of inferencing steps and docker images, Validatrr is faster for small RDF graphs. Validatrr is about an order of magnitude faster until 10,000 triples, namely, 1-2s per RDF graph compared to 30s per RDF graph for RDFUnit. After 100,000 triples, Validatrr is slower than RDFUnit, as Validatrr's linearly growing execution time surpasses RDFUnit's execution time.

Custom inferencing steps' performance impact We compare the execution time of Validatrr to RDFUnit when using a custom set of inferencing steps. We use RDFS entailment (ρ): it is commonly used, and the evaluation of Section 6.2 showed it affects the number of violations found. For Validatrr, we include the RDFS rules during validation. For RDFUnit, we include an RDFS entailment preprocessing step, as RDFUnit's docker image does not allow configuration to use a SPARQL engine that has inferencing capabilities. However, even if it would be possible to use a different SPARQL engine, a reasoning preprocessing step would still be needed for use cases that require support for a specific set of inferencing steps, not covered by typical entailment regimes [1].

To keep the measures comparable, we use the EYE reasoner as used in Validatrr with the same RDFS entailment rule set to execute the reasoning preprocessing step. This also precludes the need to compare with other sets of inferencing steps than RDFS entailment: the conclusions will be similar due to the usage of the same reasoner. Fig. 4 depicts the timings of RDFUnit and Validatrr. For RDFUnit, it depicts the combined timings of RDFS entailment as preprocessing step and validation on the newly inferred RDF graph(RDFUnit (ρ)), and it depicts solely the validation timings on the newly inferred graph (RDFUnit). For Validatrr, it depicts the timings of the validation with the two sets of inferencing rules (Validatrr (ρ) and Validatrr (υ), respectively).

Validatrr's performance is not affected by using a different set of inferencing steps, whereas the preprocessing step deteriorates RDFUnit's performance. This effect is noticable starting from RDF graphs of 10,000 triples. For RDF graphs of one million triples, compared to the previous evaluation, median execution time rises from 27s to 210s for RDFUnit, largely due to the reasoning preprocessing step.

The number of found violations inversely affects the validation execution speed. Most original violations handle missing domain and range classes, which is inferred in RDFS entailment. Statistical analysis does not allow us to accept the null hypothesis that the number of violations found is inversely correlated to the execution time. However, we notice increased performance for both approaches when less violations need to be handled. Compared to previous evaluation, for one million triples, execution time (without reasoning preprocessing) drops from 27s to 21s for RDFUnit, and from 116s to 80s for Validatrr.

The performance evaluations show that the execution time of Validatrr outperforms RDFUnit for small RDF graphs up to 100,000 triples, and its linear scaling behavior is not affected by including RDFS entailment during validation. Validatrr outperforms RDFUnit when reasoning preprocessing is needed, i.e., when the used SPARQL endpoint does not support inferencing up to the needed expressiveness, or cannot be sufficiently customized to the use case. Where RDFUnit first needs to infer all implicit data before valida-

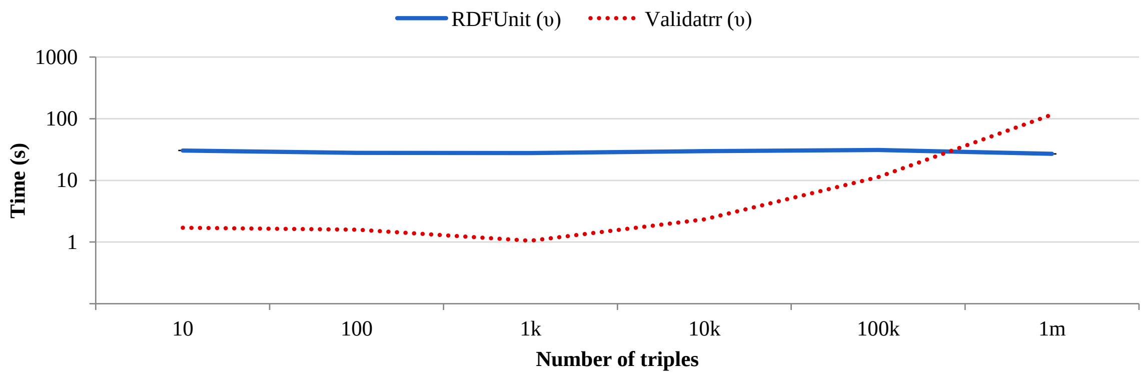

Figure 3. Validatrr's execution speed (dotted line) is up to an order of magnitude faster than RDFUnits's (solid line) when the number of triples per RDF graph is below 100,000 triples

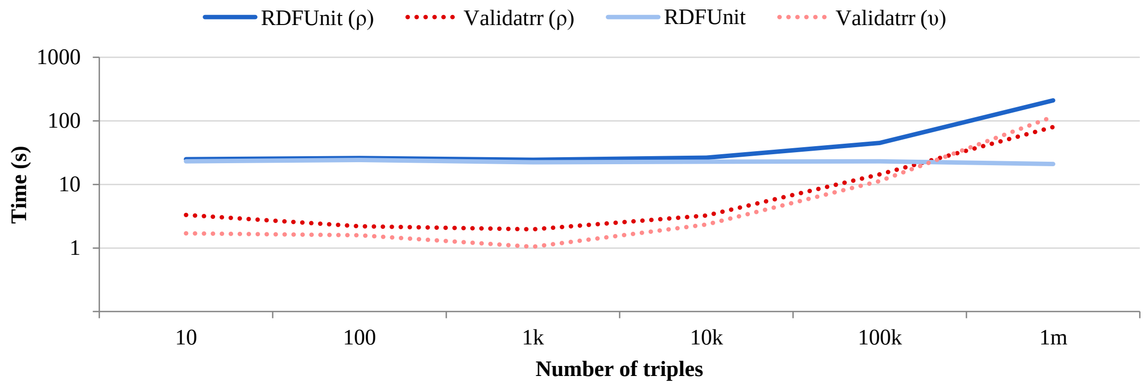

Figure 4. Validatrr's performance is not affected when including the RDFS inferencing rules (dotted line, compared to the lighter dotted line), whereas the reasoning preprocessing time deteriorated RDFUnit's performance (solid line, compared to the lighter solid line).

tion, Validatrr can infer this data during validation, and thus performs better. We thus accept Hypothesis 4.

7. Conclusion and future work

In this section, we discuss our proposed rule-based reasoning validation approach and introduced implementation. We provide concluding remarks and guide towards future work with respect to (i) the detailed root cause explanations, (ii) the fine-grained level of configuration, (iii) the number of constraint types supported by our approach, and (iv) the scaling behavior of Validatrr's performance. We close by providing some further research perspectives.

The logical proof of a validation execution, generated by the rule-based reasoner, provides a more detailed root cause explanation of why a violation occurs than the state of the art. Our evaluation does not imply that existing approaches and implementations are not capable of providing a similar level of detail. However, it does show the feasibility of more detailed explanations, and the capability of our approach to generate them. To improve the level of detail of explanations provided in the validation report, our work can guide future iterations of, e.g., SHACL's validation report descriptions, and the algorithms that generate them.

Our approach is fully configurable by adjusting different rule sets: only a single declaration and single implementation is needed to support different constraint languages, sets of inferencing steps, and validation report descriptions. This level of control considerably increases expressiveness and complexity of the validator, and a small change in a rule set could have large effects on the validation results. However, such fine-grained configuration is not needed for every use case. Future work requires investigation into configuration defaults for, a.o., ShEx and SHACL: to what extend can Validatrr be configured to function as a compliant ShEx or SHACL validator, and how will the combination of inferencing rule sets look like? A short-term goal is showing that Validatrr with the right configuration passes the core SHACL tests and is included as a compliant SHACL validator in the respective W3C documentation[18]. As such, we can provide a compliant SHACL validator where `sh:entailment` is accurately supported: the user can choose exactly which inferencing rule set is supported during validation, and can choose not to rely on the predefined custom set of inferencing steps (i.e., support for `rdfs:subClassOf`, but no other RDFS entailments) as currently specified in SHACL [47].

Our approach supports an equivalent number of constraint types compared to the state of the art, with description logic-expressiveness up to at least OWL-RL. An important point of interest is handling recursion, one of the main differences between ShEx and SHACL. The semantics of ShEx are defined, also for recursion [10], and – as it is currently undefined in the SHACL specification [47] – current works are investigating recursion in combination with negation for SHACL [19]. Future work for our approach is investigating recursion, taking into account the conclusions and mentioned complexity issues of aforementioned works. Accepting that the general problem is NP-Hard, using rule-based reasoning gives us a strong tool to handle recursion. A rule-based reasoner such as the EYE reasoner has path detection: different validations calling each other can be handled, as path detection prevents the reasoner from applying the same rule to the same data twice. In this regard, we can further investigate whether the strategies of Answer Set Programming [27] help to solve related problems, taking into account their two kinds of negation (Negation as Failure and strong negation). After investigating which rules are needed to handle recursion, the user can choose whether or not recursion should be supported during validation, as these extra rules can be added or not.

The performance of Validatrr is up to an order of magnitude faster than RDFUnit for RDF graphs up to 100,000 triples, and scales linearly w.r.t. the number of triples in the RDF graph. However, it scales less than RDFUnit, making Validatrr less suitable for large RDF graphs. As such, a trade-off must be made: our approach, which performs better for smaller RDF graphs, allows fine-grained configuration and detailed explanation, whereas other approaches scale better but do not provide the same level of detail. For future work, further investigation into related works that aim to improve the performance of rule-based reasoners, such as the work of Arndt et al. [2], can be used to improve the current scaling behavior of Validatrr.

Further research perspectives include validation of RDF graph generation descriptions, and automatic graph refinement based on violation explanations. The combination reduces the effort required to provide high-quality RDF graph generation descriptions, and is being further investigated by Heyvaert et al. [38].

On the one hand, a declarative description for generating an RDF graph – e.g., using the RDF Mapping Language (RML) [25] – can be validated, to show whether that description produces a valid RDF graph [26]. Certain constraints that apply to the description can be inferred based on the constraints that apply to the RDF graph. By including a custom inferencing rule set that reflects such inferencing in Validatrr, the generation description can be validated based on the set of constraints that apply to the RDF graph. As such, only a single set of constraints needs to be maintained and understood. The requirements of this custom inferencing rule set, and which constraint types can be applied to generation descriptions, is future work.

On the other hand, rules that handle the accurate explanations of why a violation is returned, can provide suggestions to (automatically) resolve the violation. For example, the constraint specifying "every book should have either an ISSN or an ISBN number" is violated by a resource that has both numbers. Suggestions include removing the ISSN number and removing the ISBN number. Which types of suggestions can be provided, and in which order these should be applied, is future work.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (VLAIO), and the European Union. Ruben Verborgh is a postdoctoral fellow of the Research Foundation – Flanders (FWO).

Appendix A. Updated Constraint Types Coverage

[18]https://w3c.github.io/data-shapes/data-shapes-test-suite/

Table 4

Coverage of validation approaches w.r.t. constraint types. Taken from Hartmann et al. [36], updated take the recent advancements of SHACL into account [47, 49]. Changes w.r.t. the original tabe of Hartmann et al. are marked grey.

Type	Name	SHACL	Validatrr
A01	*Functional Properties	✓	✓
A02	*Inverse-Functional Properties	✗	✓
A03	*Primary Key Properties	✗	✓
A04	*Subsumption	✓	✓
A05	*Sub-Properties	✗	✓
A06	*Object Property Paths	✗	✓
A07	Allowed Values	✓	✓
A08	Not Allowed Values	~	✓
A09	*Class Equivalence	✗	✓
A10	*Equivalent Properties	✓	✓
A11	Literal Value Comparison	✓	✓
A12	Value is Valid for Datatype	✓	✓
A13	*Property Domains	✓	✓
A14	*Property Ranges	✓	✓
A15	*Class-Specific Property Range	✓	✓
A16	Data Property Facets	✓	✓
A17	Literal Ranges	✓	✓
A18	Negative Literal Ranges	~	✓
A19	IRI Pattern Matching	✓	✓
A20	Literal Pattern Matching	✓	✓
A21	Negative Literal Pattern Matching	~	✓
A22	*Existential Quantifications	✓	✓
A23	*Universal Quantifications	✓	✓
A24	*Value Restrictions	✓	✓
A25	Use Sub-Super Relations in Validation	✗	✓
A26	Negative Property Constraints	~	✓
A27	Language Tag Matching	✓	✓
A28	Language Tag Card.	~	✓
A29	Whitespace Handling	✗	✗
A30	HTML Handling	✗	✗
A31	Structure	✓	✗
A32	*Minimum Unqualified Card.	✓	✓
A33	*Minimum Qualified Card.	✓	✓
A34	*Maximum Unqualified Card.	✓	✓
A35	*Maximum Qualified Card.	✓	✓
A36	*Exact Unqualified Card.	✓	✓
A37	*Exact Qualified Card.	✓	✓
A38	*Cardinality Shortcuts	~	~
A39	Vocabulary	✓	✓
A40	Provenance	✓	~
A41	Required Properties	~	✓
A42	Optional Properties	~	✓

Table 5

Coverage of validation approaches w.r.t. constraint types. Taken from Hartmann et al. [36], updated take the recent advancements of SHACL into account [47, 49]. Changes w.r.t. the original tabe of Hartmann et al. are marked grey (2).

Type	Name	SHACL	Validatrr
A43	Repeatable Properties	~	✓
A44	Conditional Properties	✓	✓
A45	Recommended Properties	✓	✓
A46	Severity Levels	✓	~
A47	Labeling and Documentation	✓	~
A48	Context-Sp. Property Groups	✓	✓
A49	Context-Sp. Exclusive OR of P.	✓	~
A50	Context-Sp. Exclusive OR of P. Groups	✓	~
A51	Context-Sp. Inclusive OR of P.	✓	✓
A52	Context-Sp. Inclusive OR of P. Groups	✓	✓
A53	Mathematical Operations	~	~
A54	Ordering	✓	~
A55	*Inverse Object Properties	✓	~
A56	*Symmetric Object Properties	✓	~
A57	*Asymmetric Object Properties	✓	✓
A58	*Transitive Object Properties	✗	~
A59	*Self Restrictions	✓	~
A60	Valid Identifiers	✗	✗
A61	Recursive Queries	✓	~
A62	*Reflexive Object Properties	✓	~
A63	*Class-Sp. Reflexive Object P.	✓	~
A64	*Irreflexive Object Properties	✓	✓
A65	*Class-Specific Irreflexive Object Properties	✓	~
A66	Data Model Consistency	✓	✗
A67	Handle RDF Collections	✓	~
A68	Membership in Controlled Vocabularies	✓	~
A69	Disjoint Properties	✗	✓
A70	Disjoint Classes	~	✓
A71	String Operations	~	~
A72	Aggregations	✓	~
A73	*Individual Equality	✓	~
A74	Individual Inequality	✓	~
A75	Context-Specific Valid Classes	✗	~
A76	Context-Specific Valid Properties	✓	~ ~
A77	Property Assertions	~	~
A78	*Intersection	✓	✓
A79	*Disjunction	✓	~
A80	*Negation	✓	✓
A81	*Default Values	✓	✓

References

[1] Carlos Buil Aranda, Olivier Corby, Souripriya Das, Lee Feigenbaum, Paula Gearon, Birte Glimm, Steve Harris, Sandro Hawke, Ivan Herman, Nicholas Humfrey, Nico Michaelis, Chimezie Ogbuji, Matthew Perry, Alexandre Passant, Axel Polleres, Eric Prud'hommeaux, Andy Seaborne, and Gregory Todd Williams. SPARQL 1.1 Overview. Recommendation, World Wide Web Consortium (W3C), March 2013. URL http://www.w3.org/TR/sparql11-overview/.

[2] Dörthe Arndt, Ben De Meester, Pieter Bonte, Jeroen Schaballie, Jabran Bhatti, Wim Dereuddre, Ruben Verborgh, Femke Ongenae, Filip De Turck, Rik Van de Walle, and Erik Mannens. Improving OWL RL reasoning in N3 by using specialized rules. In Valentina Tamma, Mauro Dragoni, Rafael Gonçalves, and Agnieszka Ławrynowicz, editors, *Ontology Engineering: 12th International Experiences and Directions Workshop on OWL*, volume 9557 of *Lecture Notes in Computer Science*, pages 93–104. Springer, April 2016.

[3] Dörthe Arndt, Ben De Meester, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. Using rule based reasoning for RDF validation. In *RuleML+RR*, 2017.

[4] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. RDF 1.1 Turtle – Terse RDF Triple Language. Recommendation, World Wide Web Consortium (W3C), February 2014. URL http://www.w3.org/TR/turtle/.

[5] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data. In *Proceedings of the 13th International Semantic Web Conference*, pages 213–228. Springer, Springer International Publishing, 2014.

[6] Tim Berners-Lee. Cwm, October 2000. URL http://www.w3.org/2000/10/swap/doc/cwm.html.

[7] Tim Berners-Lee. Notation 3 Logic, August 2005. URL http://www.w3.org/DesignIssues/N3Logic.

[8] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, 8(3): 249–269, 2008.

[9] Christian Bizer and Richard Cyganiak. Quality-driven information filtering using the WIQA policy framework. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):1–10, January 2009.

[10] Iovka Boneva, Jose Emilio Labra Gayo, and Eric Prud'hommeaux. Semantics and validation of shapes schemas for rdf. In Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin, editors, *The Semantic Web – ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I*, volume 10587, pages 104–120, Cham, October 2017. Springer International Publishing.

[11] Thomas Bosch and Kai Eckert. Requirements on RDF constraint formulation and validation. In *Proceedings of the 2014 International Conference on Dublin Core and Metadata Applications*, number September 2013, pages 95–108. Citeseer, 2014.

[12] Thomas Bosch and Kai Eckert. Towards Description Set Profiles for RDF using SPARQL as Intermediate Language. *International Conference on Dublin Core and Metadata Applications*, (November 2013):129–137, 2014.

[13] Thomas Bosch, Erman Acar, Andreas Nolle, and Kai Eckert. The role of reasoning for RDF validation. In *Proceedings of the 11th International Conference on Semantic Systems*, pages 33–40. ACM, 2015.

[14] Thomas Bosch, Andreas Nolle, Erman Acar, and Kai Eckert. RDF Validation Requirements – Evaluation and Logical Underpinning. *arXiv preprint arXiv:1501.03933*, 2015.

[15] Bojan Bozic, Rob Brennan, Kevin Feeney, and Gavin Mendel-Gleason. Describing reasoning results with rvo, the reasoning violations ontology. In *MEPDaW/LDQ@ ESWC*, pages 62–69, 2016.

[16] Dan Brickley and R. V. Guha. RDF Schema 1.1. Recommendation, World Wide Web Consortium (W3C), February 2014. URL http://www.w3.org/TR/rdf-schema/.

[17] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.99. Namespace document, January 2014. URL http://xmlns.com/foaf/spec/.

[18] Fabricio Chalub and Alexandre Rademaker. Verifying integrity constraints of a rdf-based wordnet. In *Global WordNet Conference*, page 309, 2016.

[19] Julien Corman, Juan L. Reutter, and Ognjen Savković. Semantics and Validation of Recursive SHACL. In Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web – ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II*, volume 11137 of *Lecture Notes in Computer Science*, pages 318–336. Springer, Cham, 2018.

[20] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium (W3C), February 2014. URL http://www.w3.org/TR/rdf11-concepts/.

[21] Carlos Viegas Damásio, Anastasia Analyti, Grigoris Antoniou, and Gerd Wagner. Supporting open and closed world reasoning on the web. In *Principles and Practice of Semantic Web Reasoning*, pages 149–163. Springer Berlin Heidelberg, 2006.

[22] Jeremy Debattista, SÖren Auer, and Christoph Lange. Luzzu – a methodology and framework for linked data quality assessment. *J. Data and Information Quality*, 8(1):4:1–4:32, October 2016.

[23] Jeremy Debattista, Makx Dekkers, Christophe Guéret, Deirdre Lee, Nandana Mihindukulasooriya, and Amrapali Zaveri. Data on the web best practices: Data quality vocabulary. Working group note, World Wide Web Consortium, December 2016. URL https://www.w3.org/TR/vocab-dqv/.

[24] Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. Comparison of Reasoners for Large Ontologies in the OWL 2 EL Profile. *Semantic Web Journal*, 2(2):71–87, April 2011.

[25] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proceedings*. CEUR, 2014.

[26] Anastasia Dimou, Dimitris Kontokostas, Markus Freudenberg, Ruben Verborgh, Jens Lehmann, Erik Mannens, Sebastian Hellmann, and Rik Van de Walle. Assessing and refining mappings to RDF to improve dataset quality. In Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin,

Krishnaprasad Thirunarayan, and Steffen Staab, editors, *The Semantic Web – ISWC 2015*, volume 9367 of *Lecture Notes in Computer Science*, pages 133–149, Bethlehem, PA, USA, October 2015. Springer International Publishing.

[27] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer Set Programming: A Primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems*, volume 5689 of *Lecture Notes in Computer Scinece*, pages 40–110. Springer Berlin Heidelberg, 2009.

[28] Mohammed Ben Ellefi, Zohra Bellahsene, John Breslin, Elena Demidova, Stefan Dietze, Julian Szymanski, and Konstantin Todorov. Rdf dataset profiling - a survey of features, methods, vocabularies and applications. *Semantic Web Journal*, 2017.

[29] Mina Farid, Alexandra Roatis, Ihab F Ilyas, Hella-Franziska Hoffmann, and Xu Chu. Clams: bringing quality to data lakes. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2089–2092. ACM, 2016.

[30] Peter M Fischer, Georg Lausen, Alexander Schätzle, and Michael Schmidt. Rdf constraint checking. In Peter M Fischer, Gustavo Alonso, Marcelo Arenas, and Floris Geerts, editors, *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT 2015)*, volume 1330 of *CEUR Workshop Proceedings*, pages 2015–2012, Brussels, Belgium, March 2015. CEUR-WS.org.

[31] Charles L. Forgy. Rete: A fast algorithm for the many pattern/-many object pattern match problem. *Artificial Intelligence*, 19 (1):17–37, sep 1982.

[32] Birte Glimm and Chimezie Ogbuji. SPARQL 1.1 Entailment Regimes. Recommendation, World Wide Web Consortium (W3C), March 2013. URL https://www.w3.org/TR/sparql11-entailment/.

[33] Peter Haase and Guilin Qi. An Analysis of Approaches to Resolving Inconsistencies in DL-based Ontologies. In *Proceedings of the International Workshop on Ontology Dynamics (IWOD-07)*, pages 97–109, 2007.

[34] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. Recommendation, World Wide Web Consortium (W3C), March 2013. URL https://www.w3.org/TR/sparql11-query/.

[35] Thomas Hartmann. *Validation Framework for RDF-based Constraint Languages*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2016.

[36] Thomas Hartmann. Validation framework for rdf-based constraint languages - phd thesis appendix. Technical report, Karlsruher Institut für Technologie (KIT), 2016. URL http://digbib.ubka.uni-karlsruhe.de/volltexte/1000054062.

[37] Patrik J. Hayes and Peter F. Patel-Schneider. RDF 1.1 Semantics. Recommendation, World Wide Web Consortium (W3C), February 2014. URL http://www.w3.org/TR/rdf11-mt/.

[38] Pieter Heyvaert, Anastasia Dimou, Ben De Meester, and Ruben Verborgh. Rule-driven inconsistency resolution for knowledge graph generation rules. *Semantic Web Journal*, February 2019.

[39] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language – Primer (Second Edition). Recommendation, World Wide Web Consortium (W3C), December 2012. URL http://www.w3.org/TR/owl2-primer/.

[40] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the pedantic web. In *3rd International Workshop on Linked Data on the Web*, volume 628 of *CEUR Workshop Proceedings*. CEUR, 2010.

[41] Aidan Hogan, JüRgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of linked data conformance. *Web Semant.*, 14:14–44, July 2012.

[42] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. Member submission, World Wide Web Consortium (W3C), May 2004. URL https://www.w3.org/Submission/SWRL/.

[43] Michael Kifer. Rule interchange format: The framework. In Diego Calvanese and Georg Lausen, editors, *RR 2008: Web Reasoning and Rule Systems*, volume 5341 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin Heidelberg, 2008.

[44] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, jul 1995.

[45] Michael Kifer, Jos de Bruijn, Harold Boley, and Dieter Fensel. A realistic architecture for the semantic web. In Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors, *Rules and Rule Markup Languages for the Semantic Web*, volume 3791 of *Lecture Notes in Computer Science*, pages 17–29. Springer Berlin Heidelberg, 2005.

[46] Holger Knublauch. OWL 2 RL in SPARQL. Documentation, TopBraid. URL http://topbraid.org/spin/owlrl-all.html.

[47] Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL). Recommendation, World Wide Web Consortium (W3C), July 2017. URL https://www.w3.org/TR/shacl/.

[48] Holger Knublauch, James A. Hendler, and Kingsley Idehen. SPIN – Overview and Motivation. Member submission, World Wide Web Consortium (W3C), February 2011. URL https://www.w3.org/Submission/spin-overview/.

[49] Holger Knublauch, Dean Allemand, and Simon Steyskal. SHACL Advanced Features. Working group note, World Wide Web Consortium (W3C), June 2017. URL https://www.w3.org/TR/shacl-af/.

[50] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd international conference on World Wide Web*, pages 747–757. ACM, March 2014.

[51] Jose Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitris Kontokostas. *Validating RDF Data*, volume 7. Morgan & Claypool Publishers LLC, sep 2017.

[52] Jose Emilio Labra Gayo, Eric Prud'hommeaux, Harold Solbrig, and Iovka Boneva. Validating and describing linked data portals using shapes. *arXiv preprint arXiv:1701.08924*, 2017.

[53] Pablo N Mendes, Hannes Mühleisen, and Christian Bizer. Sieve: linked data quality assessment and fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 116–123. ACM, 2012.

[54] Boris Motik, Ian Horrocks, and Ulrike Sattler. Bridging the gap between owl and relational databases. In *Proceedings of WWW 2007*, volume 7, pages 74–89. Elsevier, 2009.

[55] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles (Second Edition). Recommendation, World

Wide Web Consortium (W3C), December 2012. URL https://www.w3.org/TR/owl2-profiles/.

[56] Mark A. Musen. The protégé project: A look back and a look forward. *AI Matters*, 1(4):4–12, jun 2015.

[57] Mikael Nilsson. Description set profiles: A constraint language for dublin core application profiles. Working draft, Dublin Core Metadata Initiative (DCMI), 2008. URL http://dublincore.org/documents/2008/03/31/dc-dsp/.

[58] Bijan Parsia, Nicolas Matentzoglu, Rafael Gonçalves, Birte Glimm, and Andreas Steigmiller. The owl reasoner evaluation (ore) 2015 competition report. In Thorsten Liebig and Achille Fokoue, editors, *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015)*, volume 1457 of *CEUR Workshop Proceedings*, pages 2–15, Bethlehem, PA, USA, October 2015. CEUR-WS.org.

[59] Adrian Paschke. Rules and logic programming for the web. In *Reasoning Web. Semantic Technologies for the Web of Data*, pages 326–381. Springer Berlin Heidelberg, 2011.

[60] Peter F. Patel-Schneider. Using Description Logics for RDF constraint checking and closed-world recognition. *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2014.

[61] Peter F. Patel-Schneider. Diverging Views of SHACL, October 2016. URL https://research.nuance.com/diverging-views-of-shacl/.

[62] Pieter Pauwels and Sijie Zhang. Semantic rule-checking for regulation compliance checking: an overview of strategies and approaches. In Jakob Beetz, Léon van Berlo, Timo Hartmann, and Robert Amor, editors, *32rd international CIB W78 conference, Proceedings*, 2015.

[63] Héctor Pérez-Urbina, Evren Sirin, and Kendall Clark. Validating RDF with OWL integrity constraints. Technical report, LLC, 2012. URL https://www.stardog.com/docs/4.1.3/icv/icv-specification.

[64] Axel Polleres, Cristina Feier, and Andreas Harth. Rules with Contextually Scoped Negation. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications: 3rd European Semantic Web Conference, ESWC 2006 Budva, Montenegro, June 11-14, 2006 Proceedings*, pages 332–347, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[65] Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. Shape expressions: an rdf validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 32–40. ACM, 2014.

[66] Eric Prud'hommeaux, Iovka Boneva, Jose Emilio Labra Gayo, and Gregg Kellogg. Shape Expressions Language 2.1. Draft community group report, World Wide Web Consortium (W3C), November 2018. URL http://shex.io/shex-semantics/.

[67] Filip Radulovic, Nandana Mihindukulasooriya, Raúl García-Castro, and Asunción Gómez-Pérez. A comprehensive quality model for linked data. *Semantic Web*, (Preprint):1–22, 2017.

[68] Arthur G Ryman, Arnaud Le Hors, and Steve Speicher. Oslc resource shape: A language for defining constraints on linked data. *LDOW*, 996, 2013.

[69] José Hiram Soltren. Query-based database policy assurance using semantic web technologies. Master's thesis, Massachusetts Institute of Technology, 2009.

[70] Slawek Staworko, Iovka Boneva, Jose Emilio Labra Gayo, Samuel Hym, Eric Prud'hommeaux, and Harold Solbrig. Complexity and expressiveness of shex for rdf. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[71] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L McGuinness. Integrity constraints in owl. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, Atlanta, Georgia, USA, July 2010.

[72] Dominik Tomaszuk. Rdf validation: A brief survey. In *International Conference: Beyond Databases, Architectures and Structures*, pages 344–355. Springer, 2017.

[73] Ruben Verborgh and Jos De Roo. Drawing Conclusions from Linked Data on the Web: The EYE Reasoner. *IEEE Software*, 32(5):23–27, May 2015.

[74] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for linked data: A survey. *Semantic Web Journal*, 7(1):63–93, March 2015.